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).

Если для головного объекта обнаруживается дубляж имени в подчиненных объектах, то объект не создается.

Если обнаруживается дубляж имени на дереве иерархии объектов, 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 дереве иерархии по имени (метод возвращает

указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Подразумевается, что наименования объектов на дереве иерархии уникальные;

- Метод поиска объекта на ветке дерева иерархии от текущего объекта по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Подразумевается, что наименования объектов на ветке дерева иерархии уникальные;
- 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. Дубля

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

1) Объект cout, предназначенный для вывода потока данных (iostream)

2) Объект cin, предназначенный для ввода потока данных (iostream)

3) Класс cl_base:

Поля:

- readiness:

  1)Наименование – readiness

  2)Тип - int

  3)Модификатор доступа – protected

  4)Назначение - хранит в себе свойство готовности объекта

- classification:

  1)Наименование – classification

  2)Тип - int

  3)Модификатор доступа – public

  4)Назначение - хранит в себе классификацию объекта

- subordinate_objects:

  1)Наименование – subordinate_objects

  2)Тип - vector < cl_base* >

  3)Модификатор доступа – public

  4)Назначение - динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии

Функционал:

- unparse_string:

  1)Наименование – unparse_string

  2)Тип возвращаемого значения – нет

  3)Аргументы:

std::string text

std::vector<std::string>& arr

4)Модификатор доступа – public

5)Назначение – разделение строки по пробелам

- set_readiness:

  1)Наименование – set_readiness

  2)Тип возвращаемого значения – нет

  3)Аргументы:

  •int value

  4)Модификатор доступа – public

  5)Назначение – задание нового значения готовности объекта

- get_readiness:

  1)Наименование – get_readiness

  2)Тип возвращаемого значения – int

  3)Аргументы - нет

  4)Модификатор доступа – public

  5)Назначение – метод получения готовности объекта

- turn_of_downer:

  1)Наименование – turn_of_downer

  2)Тип возвращаемого значения – void

  3)Аргументы - нет

  4)Модификатор доступа – public

  5)Назначение – метод выключения всех объектов ниже текущего

- find_object_by_name_from_root:

  1)Наименование – find_object_by_name_from_root

  2)Тип возвращаемого значения – cl_base*

  3)Аргументы:

string object_name

4)Модификатор доступа – public

5)Назначение – поиск объекта, начиная с корня

- find_object_by_name_from_currentt:

1)Наименование – find_object_by_name_from_current

2)Тип возвращаемого значения – cl_base*

3)Аргументы:

string object_name

4)Модификатор доступа – public

5)Назначение – поиск объекта, начиная с теущего

- print_tree_recur:

1)Наименование – print_tree_recur

2)Тип возвращаемого значения – void

3)Аргументы:

int n

4)Модификатор доступа – public

5)Назначение – метод рекурсивного вывода дерева объектов от текущего объекта

- print_tree_recur_with_readiness:

1)Наименование – print_tree_recur_with_readiness:

2)Тип возвращаемого значения – void

3)Аргументы:

int n

4)Модификатор доступа – public

5)Назначение – метод форматированного рекурсивного вывода дерева объектов

4) Класс cl_application:

•Функционал:

- bild_tree_objects:

1)Наименование – bild_tree_objects

2)Тип возвращаемого значения – void

3)Аргументы - нет

4)Модификатор доступа – public

5)Назначение – метод построения исходного дерева иерархии объектов (конструирования моделируемой системы)

- exec_app:

1)Наименование – exec_app

2)Тип возвращаемого значения – int

3)Аргументы - нет

4)Модификатор доступа – public

5)Назначение – метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

5) Переменная целочисленного типа int

6) Базовый класс cl_base

7) Объект класса cl_base

8) Класс объекта приложения cl_application

9) Объект класса cl_application

10) Модификаторы доступа (public, protected)

11) Конструктор cl_application

12) Конструктор cl_base

13) Инкремент

14) Оператор сравнения (<, ==, !=)

15) Объект класса string (string)

16) Объект класса string (vector)

17) Метод getline (string)

18) Метод stoi (string)

19) Арифмитический оператор (+, =)

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 функции main

Функционал: Главная функция программы.

Параметры: .

Возвращаемое значение: int, Код возврата.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание главного обеъекта типа cl_application с параметром p_head_object = nullptr	2
2		Вызов метода build_tree_object у головного объекта cl_application	3
3		Вывод "Object tree\n"	4
4		Вызов метода print_tree_recur с параметром 0 у головного объекта cl_application	5
5		Вывод "The tree of objects and their readiness"	6
6		return значения, полченного вызова метода exec_app у головного объекта cl_application	∅

3.2 Алгоритм метода set_readiness класса cl_base

Функционал: Задание нового значения готовности объекта.

Параметры: int, value, новое значение для поля readiness.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_readiness класса cl_base

№	Предикат	Действия	№ перехода
1		поле readiness = value	∅

3.3 Алгоритм метода get_readiness класса cl_base

Функционал: Получение значения готовности объекта.

Параметры: .

Возвращаемое значение: int, значение поля readiness.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода get_readiness класса cl_base

№	Предикат	Действия	№ перехода
1		return значения поля readiness	∅

3.4 Алгоритм метода turn_of_downer класса cl_base

Функционал: Метод выключения всех объектов ниже текущего.

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода turn_of_downer класса cl_base

№	Предикат	Действия	№ перехода
1		Поле readiness = 0	2
2		Инициализация целочисленно перменной i нулем	3
3	i < размер массива subordinate_objects	вызов метода turn_of_downer к subordinate_objects[i]	4

№	Предикат	Действия	№ перехода
			∅
4		Инкримент i	3

3.5 Алгоритм метода unparse_string класса cl_base

Функционал: Получение значения готовности объекта.

Параметры: string, text, исходная строка; vector<std::string>&, array, массив для сохранения элементов.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода unparse_string класса cl_base

№	Предикат	Действия	№ перехода
1		string tmp = ""	2
2		Инициализация целочисленно перменной i нулем	3
3	i < размер строки text		4
			6
4	text[i] равен ' '?	Добавить tmp в массив arr tmp = ""	5
		tmp += text[i]	5
5		Инкремент i	3
6		Добавить tmp в массив arr	∅

3.6 Алгоритм метода find_object_by_name_from_root класса cl_base

Функционал: Поиск объекта, начиная с корня.

Параметры: string, object_name, имя искомого объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода find_object_by_name_from_root класса cl_base

№	Предикат	Действия	№ перехода
1		Возвращения значения полученного вызовом метода find_object_by_name_from_current к PARENTS[0] с параметром object_name	∅

3.7 Алгоритм метода find_object_by_name_from_current класса cl_base

Функционал: Поиск объекта, начиная с текущего.

Параметры: string, object_name, имя искомого объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода find_object_by_name_from_current класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленно перменной i нулем	2
2	i < размер subordinate_objects		3
			5
3	имя subordinate_objects[i] == object_name	return subordinate_objects[i]	∅
			4
4		Инкремент i	2
5		Инициализация целочисленно перменной i нулем	6
6	i < размер subordinate_objects		7
			9
7	значение полученное	return значение полученное методом	∅

№	Предикат	Действия	№ перехода
	методом find_object_by_name_from_current с параметром object_name, вызванный к subordinate_objects[i]	find_object_by_name_from_current с параметром object_name, вызванный к subordinate_objects[i]	
			8
8		Инкремент i	6
9		return nullptr	∅

3.8 Алгоритм метода print_tree_recur класса cl_base

Функционал: Метод рекурсивного вывода дерева объектов от текущего объекта.

Параметры: int, n, количество квадранта пробелов.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода print_tree_recur класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленно перменной space нулем	2
2	space < n	Вывод " " (4 пробела)	3
			4
3		Инкремент space	2
4		Вывод "(значения, полченного меодом get_name)\n"	5
5		Инициализация целочисленно перменной i нулем	6
6	i < размер массива subordinate_objects	вызов метода print_tree_recur с параметром n+1, к объекту subordinate_objects[i]	7
			∅

№	Предикат	Действия	№ перехода
7		Инкремент i	6

3.9 Алгоритм метода print_tree_recur_with_readiness класса cl_base

Функционал: Метод рекурсивного форматированного вывода дерева объектов от текущего объекта.

Параметры: int, n, количество квадранта пробелов.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода print_tree_recur_with_readiness класса cl_base

№	Предикат	Действия	№ перехода
1	значение полученное методом get_readiness не равно 0	Вывод "\n"	2
		Вывод "\n"	6
2		Инициализация целочисленно перменной space нулем	3
3	space < n	Вывод " " (4 пробела)	4
			5
4		Инкремент space	3
5		Вывод "(значение, полученное методом get_name) is ready"	10
6		Инициализация целочисленно перменной space нулем	7
7	space < n	Вывод " " (4 пробела)	8

№	Предикат	Действия	№ перехода
			9
8		Инкремент space	7
9		Вывод "(значение, полученное методом get_name) not is ready"	10
10		Инициализация целочисленно перменной i нулем	11
11	i < размер массива subordinate_objects	Вызов метода print_tree_recur_with_readiness с параметром n + 1 к объекту subordinate_objects[i]	12
			∅
12		Инкремент i	11

3.10 Алгоритм метода build_tree_objects класса cl_application

Функционал: Метод построения исходного дерева иерархии объектов (конструирования моделируемой системы).

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Создание a, b типа string	2
2		Ввод s_object_name	3
3		Инициализация flag3 значением true	4
4	flag3	Ввод a	5
			18
5	a == "endtree"	break	18

№	Предикат	Действия	№ перехода
			6
6		Ввод b, c	7
7	a == b	flag3 = false	18
		Инициализация flag1(bool) значением true Инициализация index(int) значением -1	8
8		Инициализация целочисленно перменной i нулем	9
9	i < рамзер массива PARENTS		10
			12
10	значение, полученное методом get_name к объекту PARENTS[i] == a	index = i flag1 = false break	11
			11
11		Инкремент i	9
12	flag1	Инициализация целочисленно перменной i нулем	13
			16
13	i < рамзер массива PARENTS		14
			16
14	значение метода get_subordinate_object_by_name с параметром a, к объекту PARENTS[i] != nullptr	Добавить в PARENTS PARENTS[i]->get_subordinate_object_by_name(a) index = PARENTS.size() - 1 break	15
			15
15		Инкремент i	14
16	index != -1	создание экземпляра cl_base с параметрами (PARENTS[index], b)	17

№	Предикат	Действия	№ перехода
		Поле classification объекта object задать значением c	
			17
17	значение метода get_name == a	Поле readiness = 1 PARENTS добавить текущий объект this создание экземпляра cl_base с параметрами (this, b) Поле classification = c	18
			18
18		Очиска буфера cin Ввод a посредством getline Ввод a посредством getline	19
19	flag3	Создание переменной line(string) Создание массива arr(vector<string>) Ввод line посредством getline	20
			∅
20	Длина line == 0	break	∅
			21
21		вызов меода unparse_string с параметрами (line, arr) a = arr[0] b = arr[1]	22
22	a==b	flag3 = false; break;	∅
			23
23	a == значения get_name от объекта PARENTS[0]		24
			25
24	b, привиденный к int == 0	turn_of_downer к объекту PARENTS[0]	∅
		set_readiness с параметром b, приведенной к int, к объекту PARENTS[0]	∅

№	Предикат	Действия	№ перехода
25	поле readiness от головного объекта, объекта найденного с помощью метода find_object_by_name_from_root с параметром a != 0		26
			∅
26	b, привиденный к int == 0	turn_of_downer к объекту, получпенным методом find_object_by_name_from_root с параметром a	∅
		поле readiness задачть значением b, привиденное к int, объекта, получпенного методом find_object_by_name_from_root с параметром a	∅

3.11 Алгоритм метода exec_app класса cl_application

Функционал: метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи)..

Параметры: .

Возвращаемое значение: int, Код возврата.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода print_tree_recur_with_readiness с параметром 0	2
2		return 0	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-14.

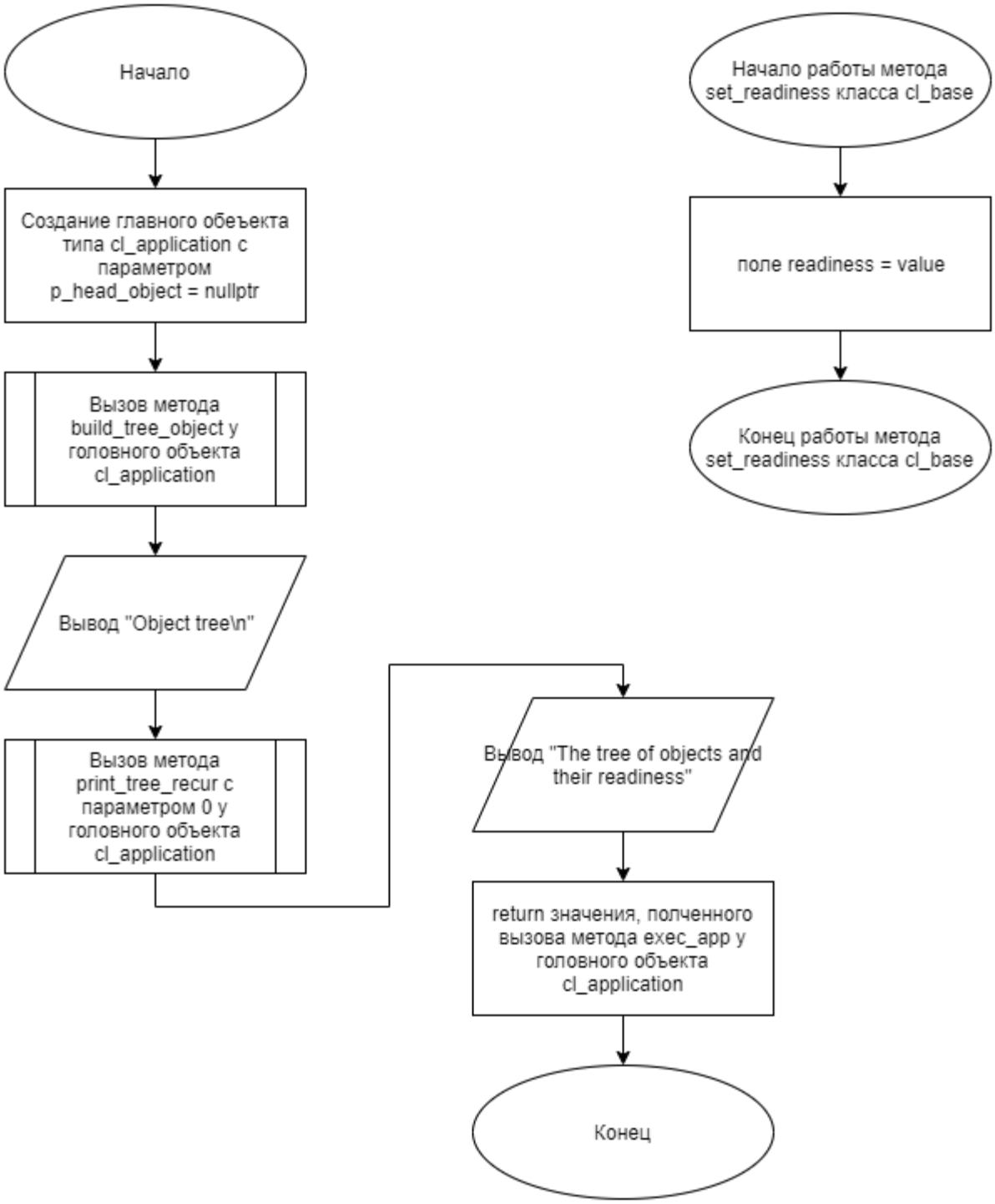

Рисунок 1 – Блок-схема алгоритма

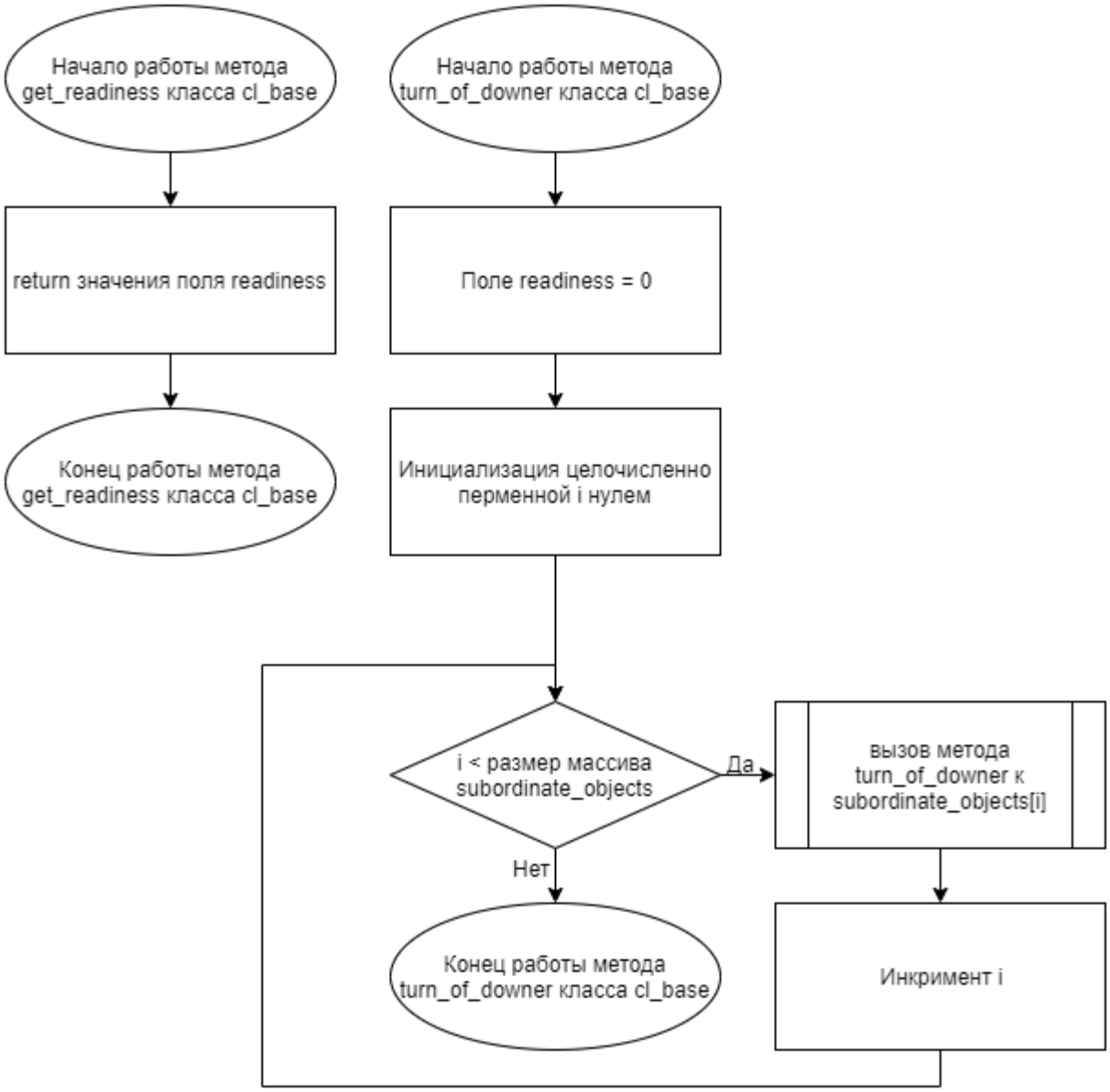

Рисунок 2 – Блок-схема алгоритма

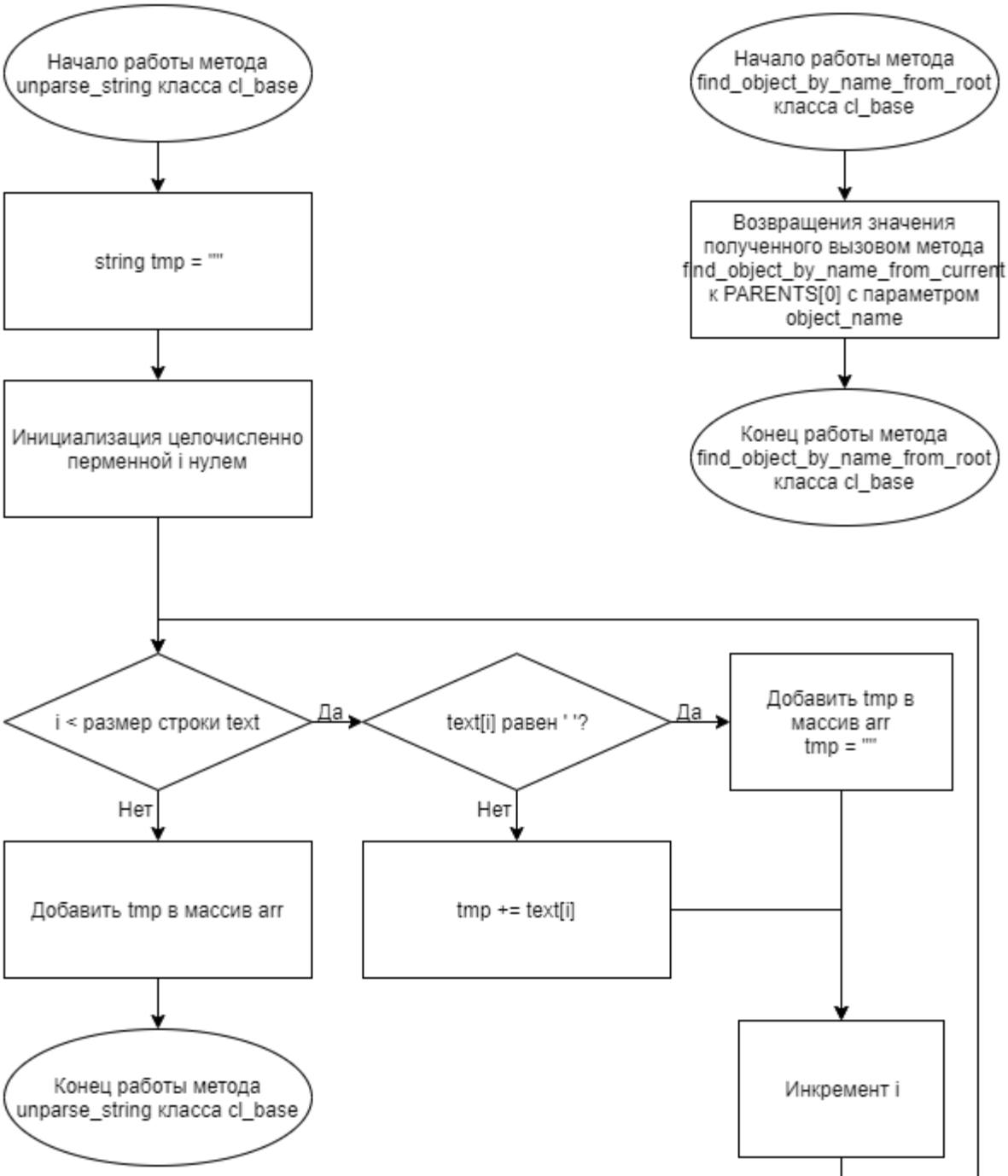

Рисунок 3 – Блок-схема алгоритма

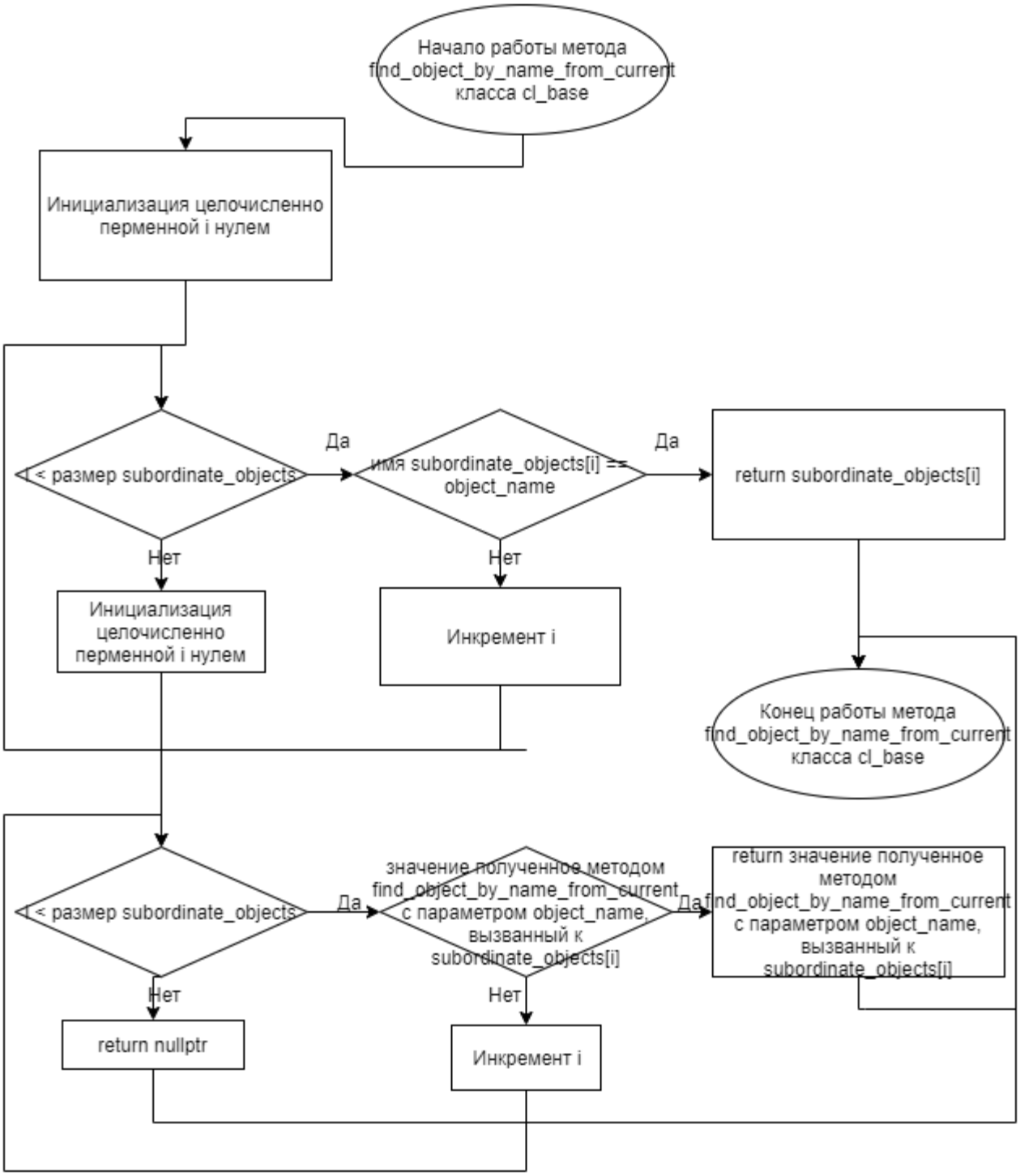

Рисунок 4 – Блок-схема алгоритма

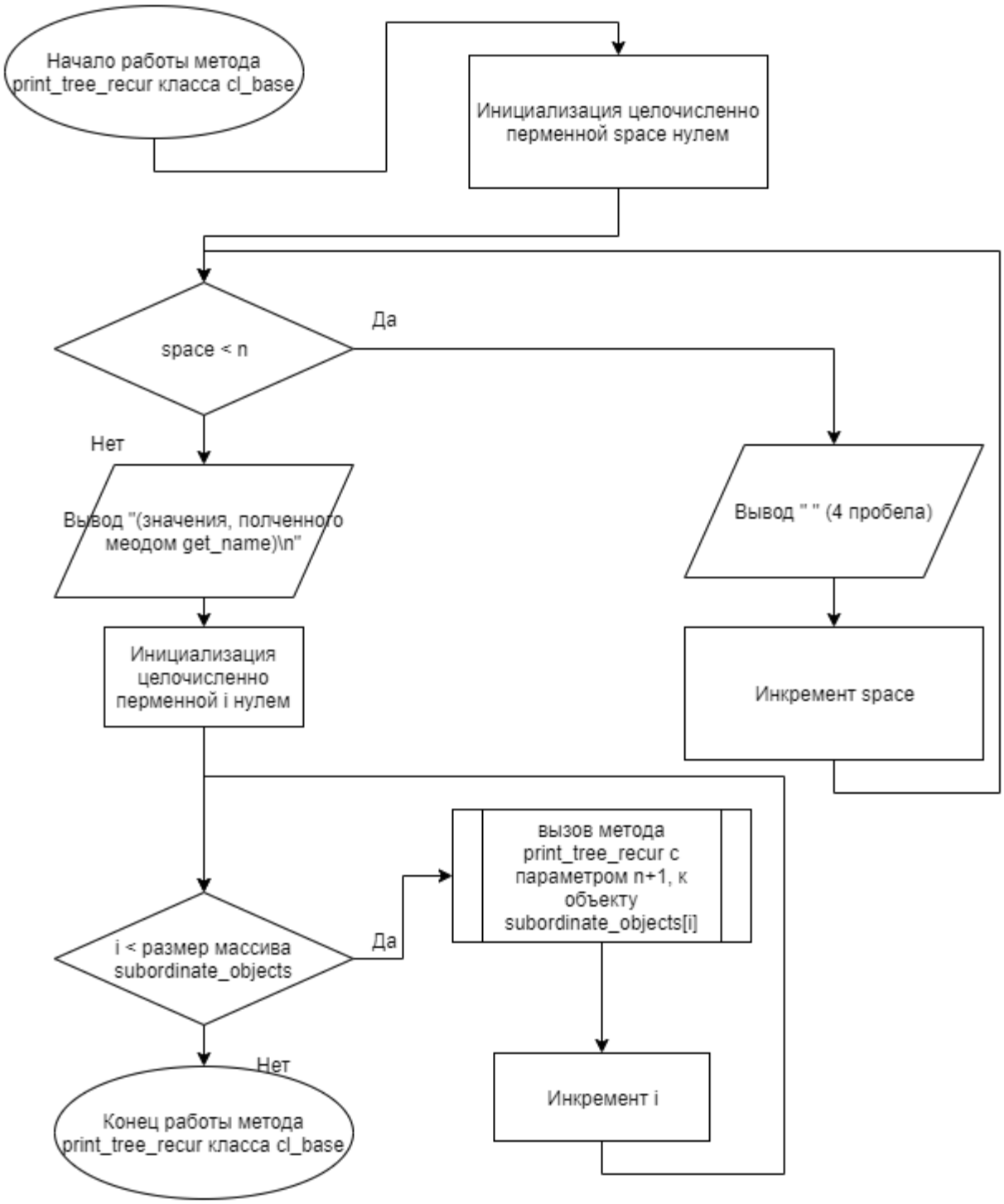

Рисунок 5 – Блок-схема алгоритма

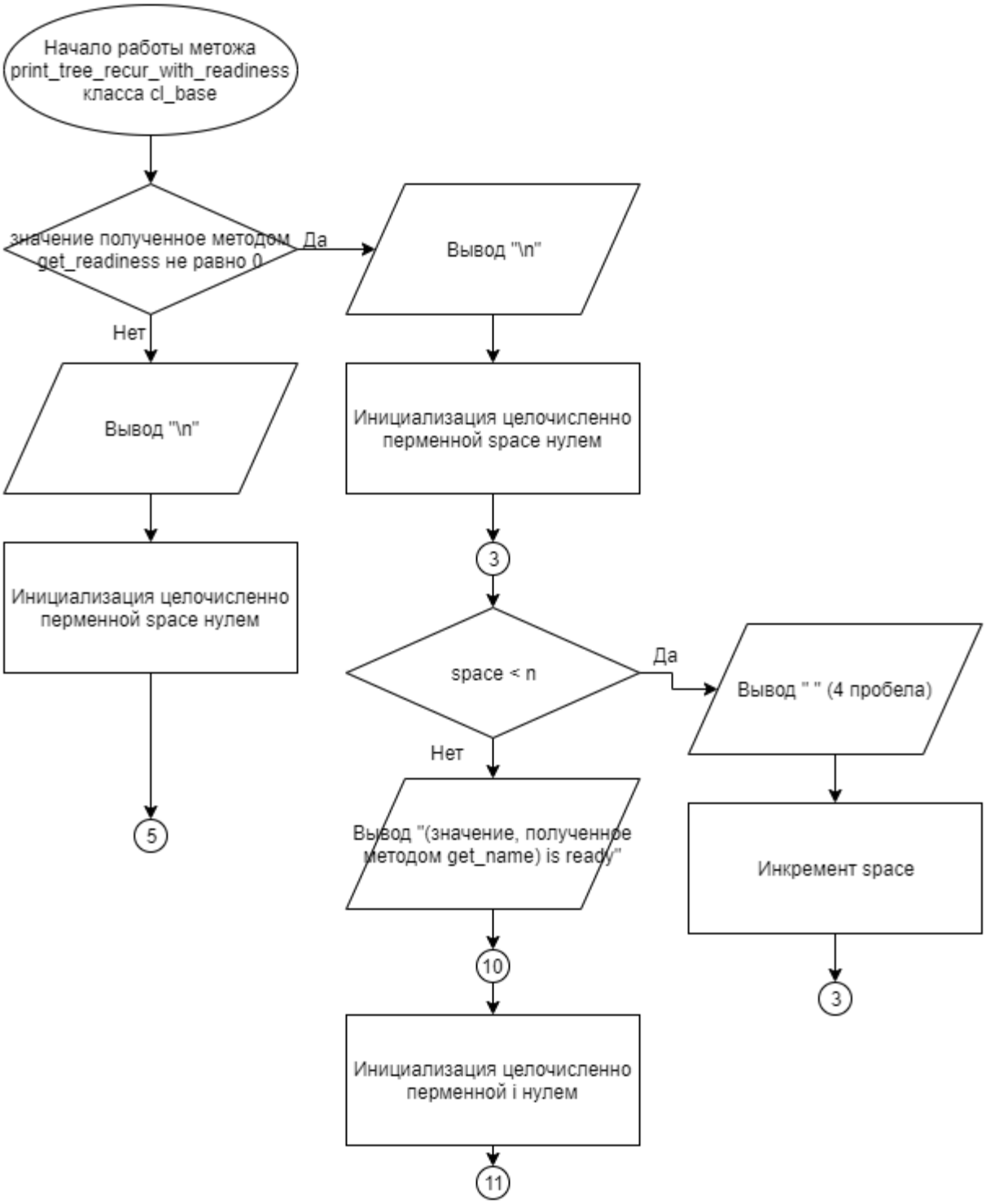

Рисунок 6 – Блок-схема алгоритма

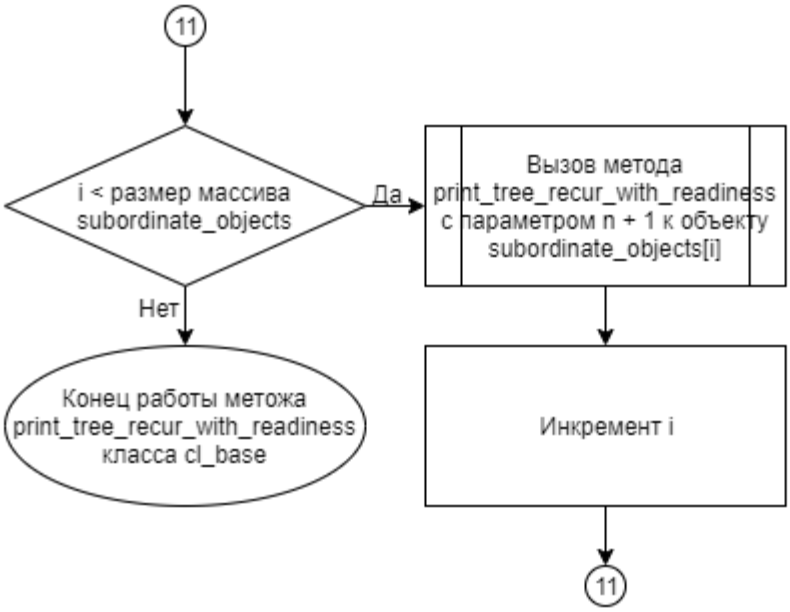

Рисунок 7 – Блок-схема алгоритма

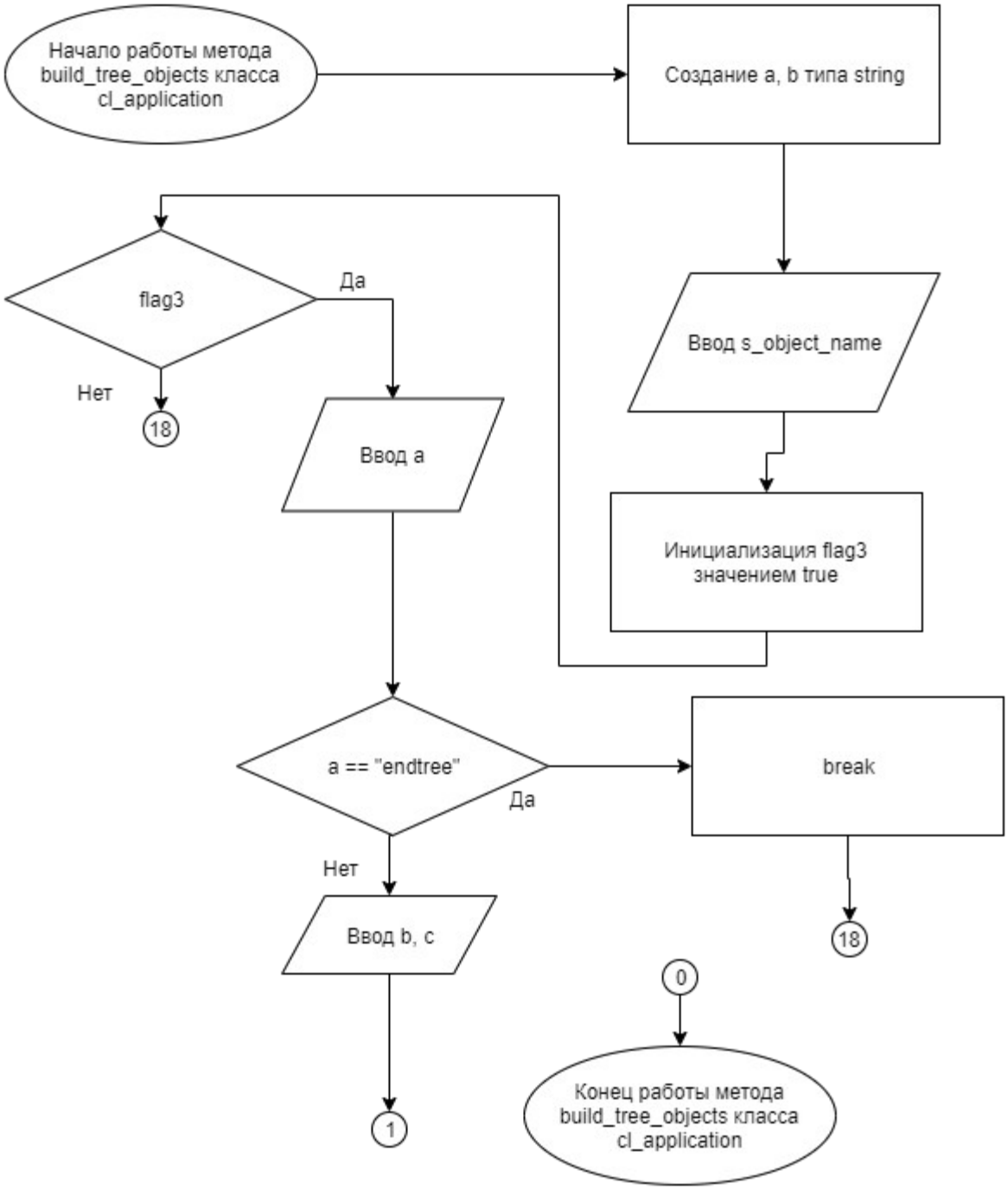

Рисунок 8 – Блок-схема алгоритма

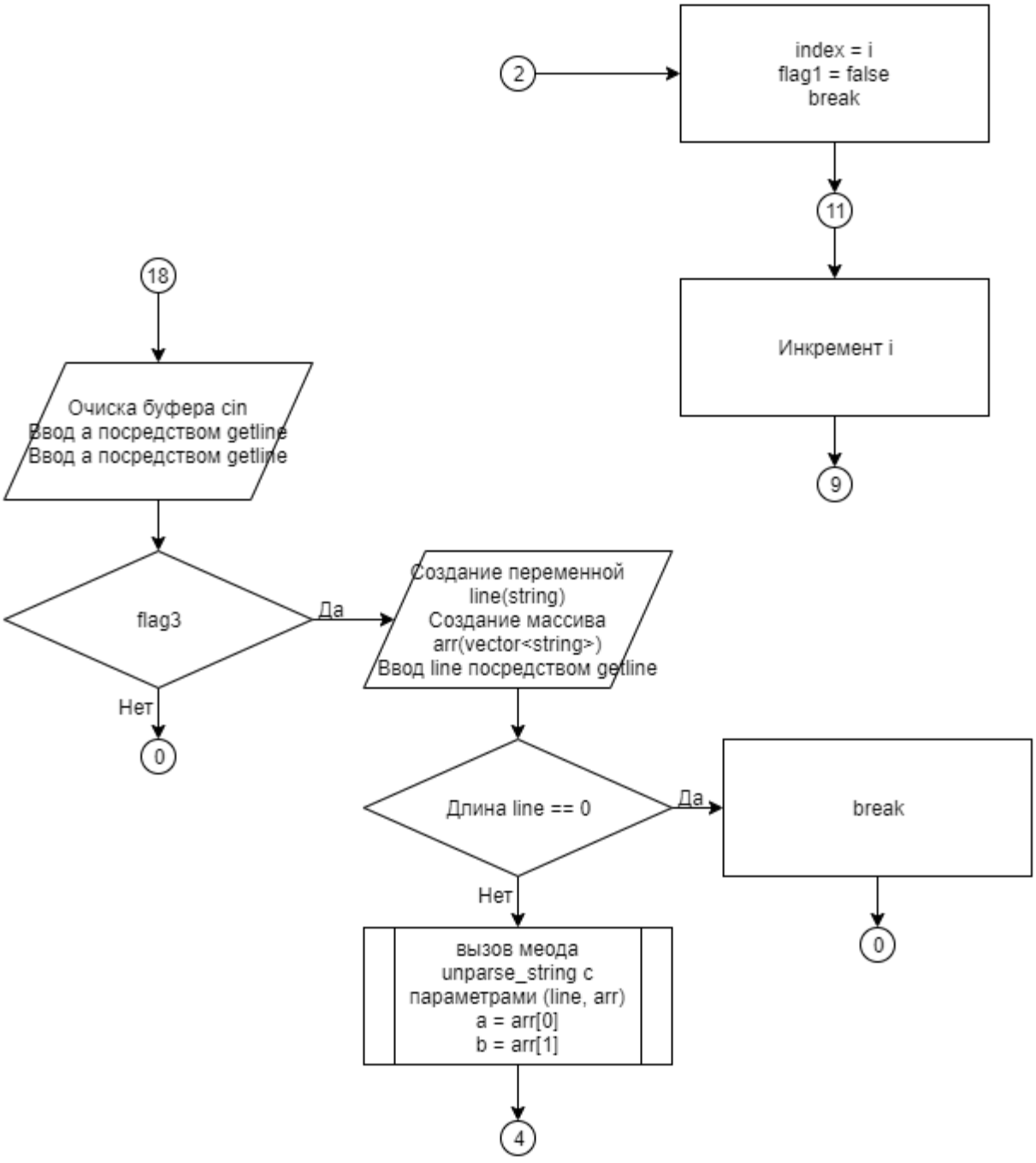

Рисунок 9 – Блок-схема алгоритма

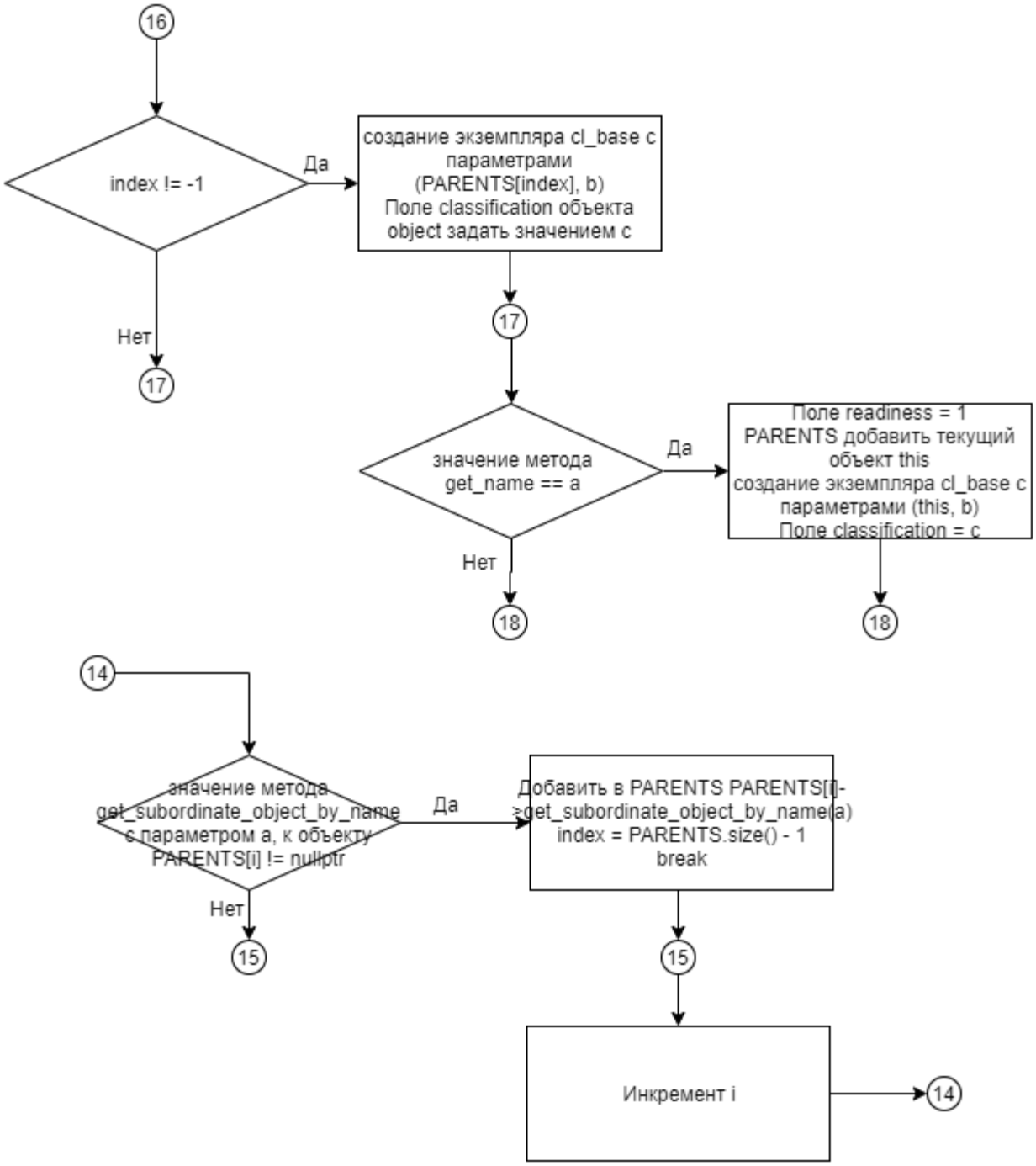

Рисунок 10 – Блок-схема алгоритма

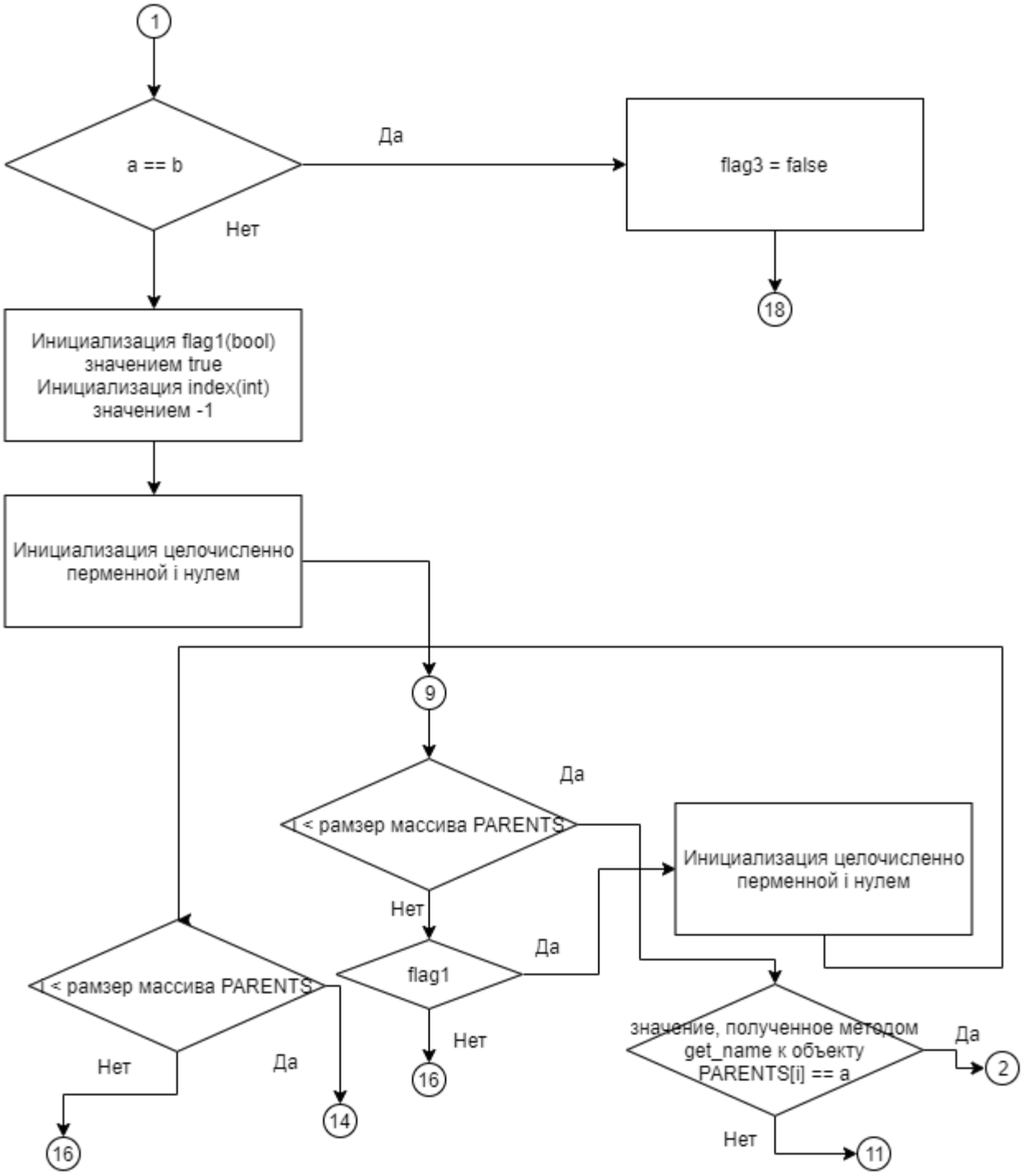

Рисунок 11 – Блок-схема алгоритма

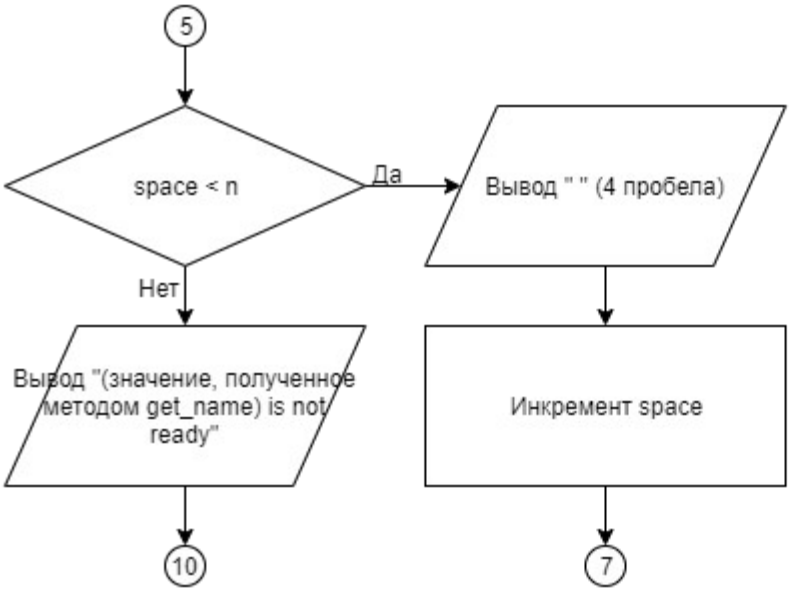

Рисунок 12 – Блок-схема алгоритма

Рисунок 13 – Блок-схема алгоритма

Рисунок 14 – 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_application*	p_head_object,	std::string
s_object_name) {
	this->p_head_object = p_head_object; // Указатель на головной объект
	this->s_object_name = s_object_name;
	if (p_head_object){
		p_head_object->subordinate_objects.push_back(this);	//	добавление	в
состав подчиненных головного объекта
	}
}
void cl_application::bild_tree_objects() {
	//Метод	построения	исходного	дерева	иерархии	объектов	(конструирования
моделируемой системы);
	std::string a, b;
	int c;
	std::cin >> s_object_name;
	bool flag3 = true;
	while (flag3) {
		std::cin >> a;
		if (a == "endtree") {
			break;
		}
		std::cin >> b >> c;
		if (a == b) {
			flag3 = false;
		}
		else {
		bool flag1 = true;
		int index = -1;
		for (int i = 0; i < PARENTS.size(); i++) {
			if (PARENTS[i]->get_name() == a) {
				index = i;
				flag1 = false;
				break;
			}
		}
		if (flag1) {
			for (int i = 0; i < PARENTS.size(); i++) {
				if	(PARENTS[i]->get_subordinate_object_by_name(a)	!=
```

```
nullptr) {
                                    PARENTS.push_back(PARENTS[i]-
>get_subordinate_object_by_name(a));
                                          index = PARENTS.size() - 1;
                                          break;
                              }
                        }
                  }
                  if (index != -1) {
                        cl_base* object = new cl_base(PARENTS[index], b);
                        object->classification = c;
                  }
                  else if (this->get_name() == a) {
                        this->readiness = 1;
                        PARENTS.push_back(this);
                        cl_base* object = new cl_base(this, b);
                        object->classification = c;
                        }
                  }
            }
            std::cin.clear();
            std::getline(std::cin, a, '\n');
            std::getline(std::cin, a, '\n');
            while (flag3) {
                  std::string line;
                  std::vector<std::string> arr;
                  std::getline(std::cin, line, '\n');
                  if (line.length() == 0) break;
                  unparse_string(line, arr);
                  a = arr[0];
                  b = arr[1];
                  if (a == b) {
                        flag3 = false;
                        break;
                  }
            else if (a == PARENTS[0]->get_name()) {
            if (std::stoi(b) == 0) {
                  PARENTS[0]->turn_of_downer();
                  }
            else {
                  PARENTS[0]->set_readiness(std::stoi(b));
                  }
            }
            else         if         (find_object_by_name_from_root(a)->get_head_object()-
>get_readiness() != 0) {
                  if (std::stoi(b) == 0) {
                        find_object_by_name_from_root(a)->turn_of_downer();
                        }
                  else {
                        find_object_by_name_from_root(a)->set_readiness(std::stoi(b));
                        }
                  }
            }
}

int cl_application::exec_app() {
      //  Метод  запуска  приложения  (начало  функционирования  системы,  выполнение
```

```
алгоритма решения задачи).
	this->print_tree_recur_with_readiness(0);
	return 0;
}
```

5.2 Файл cl_application.h

Листинг 2 – cl_application.h

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

class cl_application : public cl_base {
public:
	cl_application(cl_application*  p_head_object,  std::string  s_object_name  =
"Base_object");
	void  bild_tree_objects();  //Метод  построения  исходного  дерева  иерархии
объектов (конструирования моделируемой системы);
	int exec_app();// Метод запуска приложения (начало функционирования системы,
выполнение алгоритма решения задачи).
};
#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"


std::vector <cl_base*> PARENTS;
cl_base::cl_base(cl_base*   p_head_object,   std::string   s_object_name)   {   //
Параметризированный конструктор с параметрами;
	this->p_head_object = p_head_object;
	this->s_object_name = s_object_name;
	if (p_head_object)
	{
		p_head_object->subordinate_objects.push_back(this);
	}
}
bool cl_base::change_name_object(std::string object_name) {
	//Метод  редактирования  имени  объекта.  Один  параметр  строкового  типа,
содержит новое наименование объекта. Если нет дубляжа имени подчиненных объектов у
головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь»;
	for (int i = 0; i < PARENTS.size(); i++) {
		if   ((PARENTS[i]->get_name()   !=   object_name)   &&   (PARENTS[i]-
>get_subordinate_object_by_name(object_name) == nullptr)) {
			this->s_object_name = object_name;
```

```
                  return true;
                  }
            }
            return false;
}


cl_base* cl_base::find_object_by_name_from_root(std::string object_name) { //new
      return PARENTS[0]->find_object_by_name_from_current(object_name);
}

cl_base*   cl_base::find_object_by_name_from_current(std::string   object_name)   {
//new
      for (int i = 0; i < this->subordinate_objects.size(); i++) {
            if  (this->subordinate_objects[i]->get_name()  ==  object_name)  return
this->subordinate_objects[i];
      }

      for (int i = 0; i < this->subordinate_objects.size(); i++) {
            if                                          (this->subordinate_objects[i]-
>find_object_by_name_from_current(object_name))            return            this-
>subordinate_objects[i]->find_object_by_name_from_current(object_name);
      }
      return nullptr;
}

void cl_base::print_tree_recur(int n=0) {
      for (int space = 0; space < n; space++) std::cout << "    ";
      std::cout << this->get_name() << '\n';
      for (int i = 0; i < this->subordinate_objects.size(); i++) {
            this->subordinate_objects[i]->print_tree_recur(n + 1);
      }
}

void cl_base::print_tree_recur_with_readiness(int n = 0) {
      if (this->get_readiness() != 0) {
            std::cout << '\n';
            for (int space = 0; space < n; space++) std::cout << "    ";
            std::cout << this->get_name() << " is ready";
      }
      else {
            std::cout << '\n';
            for (int space = 0; space < n; space++) std::cout << "    ";
            std::cout << this->get_name() << " is not ready";
      }
      for (int i = 0; i < this->subordinate_objects.size(); i++) {
            this->subordinate_objects[i]->print_tree_recur_with_readiness(n + 1);
      }
}

void cl_base::turn_of_downer() {
      this->readiness = 0;
      for (int i = 0; i < this->subordinate_objects.size(); i++) {
            this->subordinate_objects[i]->turn_of_downer();
      }
}
```

```
void cl_base::set_readiness(int value) {
	this->readiness = value;
}
int cl_base::get_readiness() {
	return this->readiness;
}


std::string cl_base::get_name() {
	//Метод получения имени объекта;
	return s_object_name;
}

cl_base* cl_base::get_head_object() {
	//Метод получения указателя на головной объект текущего объекта;
	return p_head_object;
}

void cl_base::show_tree_objects() {
	//Метод вывода наименований объектов в дереве иерархии слева направо и
сверху вниз;
	for (int i = 0; i < PARENTS.size(); i++) {
		if (PARENTS[i]->get_head_object() == nullptr) {
			std::cout << PARENTS[i]->get_name() << '\n';
			break;
		}
	}
	for (int i = 0; i < PARENTS.size(); i++) {
		std::cout << PARENTS[i]->get_name() << "  ";
		for (int j = 0; j < PARENTS[i]->subordinate_objects.size(); j++) {
			if (j != PARENTS[i]->subordinate_objects.size() - 1) {
				std::cout	<<	(PARENTS[i]->subordinate_objects[j])-
>get_name() << "  ";
			}
		else std::cout << (PARENTS[i]->subordinate_objects[j])->get_name() <<
"  ";
		}
		if (i != PARENTS.size() - 1) {
			std::cout << '\n';
			}
		}
}
void cl_base::unparse_string(std::string text, std::vector<std::string>& arr) {
	std::string tmp = "";
	for (int i = 0; i < text.size(); i++) {
		if (text[i] == ' ') {
			arr.push_back(tmp);
			tmp = "";
		}
		else tmp += text[i];
	}
	arr.push_back(tmp);
}
cl_base* cl_base::get_subordinate_object_by_name(std::string object_name) {
	//Метод получения указателя на подчиненный объект по его имени
	if (object_name.size() != 0) {
		for (int i = 0; i < subordinate_objects.size(); i++) {
```

```
                  if (subordinate_objects[i]->get_name() == object_name) {
                        return subordinate_objects[i];
                  }
            }
      }
      return nullptr;
}
```

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>
class cl_base;
extern std::vector <cl_base*> PARENTS;
class cl_base {
protected:
      cl_base* p_head_object; // Указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно nullptr);
      std::string s_object_name; // Наименование объекта(строкового типа);
      int readiness = 0;
public:
      std::vector < cl_base* > subordinate_objects; // Динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии // new
      void set_readiness(int value);
      int get_readiness();
      int classification = 1;
      void turn_of_downer();
      cl_base(cl_base* p_head_object = nullptr, std::string s_object_name = "Base_object");
      cl_base* find_object_by_name_from_root(std::string object_name); //new
      cl_base* find_object_by_name_from_current(std::string object_name); //new
      void print_tree_recur(int n); //new
      void unparse_string(std::string text, std::vector<std::string>& array);
      void print_tree_recur_with_readiness(int n); //new
      bool change_name_object(std::string object_name); //Метод редактирования имени объекта. Один параметр строкового типа, содержит новое наименование объекта. Если нет дубляжа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь»;
      std::string get_name();//Метод получения имени объекта;
      cl_base* get_head_object(); //Метод получения указателя на головной объект текущего объекта;
      void show_tree_objects(); //Метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
      cl_base* get_subordinate_object_by_name(std::string object_name);
};
#endif
```

5.5 Файл main.cpp

Листинг 5 – main.cpp

```
#include "main.h"


int main()
{
	cl_application ob_cl_application(nullptr); // создание корневого
	ob_cl_application.bild_tree_objects();		// конструирование системы,
построение
	std::cout << "Object tree\n";
	ob_cl_application.print_tree_recur(0);
	std::cout << "The tree of objects and their readiness";
	return ob_cl_application.exec_app();// запуск системы
}
```

5.6 Файл main.h

Листинг 6 – main.h

```
#ifndef MAIN_H
#define MAIN_H
#include <iostream>
#include <vector>
#include <string>
#include "cl_base.h"
#include "cl_application.h"
#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 12.

Таблица 12 – 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).