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

- Класс cl_base:
  - Поля:
  - Методы:
    - print_tree
    - Метод print_status:
      - Функционал: выводит на экран состояние объекта и его подчиненных объектов в виде дерева, начиная с текущего объекта.
      - Параметры:
        - Целочисленный параметр level - определяющий отступ вывода объектов;
      - Тип возвращаемоого значения: нет;
      - Модификатор доступа: public;
    - Метод count:
      - Функционал: возвращает количество объектов с заданнфм наименованием в иерархии объектов, начиная с текущего объекта;
      - Параметры:
        - Строковой параметр s_name - наименование текущего объекта;
      - Тип возвращаемого значения: целочисленный;
      - Модификатор доступа: public;
    - Метод search_object:
      - Функционал: возвращает указатель на объект с заданным наименованием в иерархии объектов начиная с

текущего объекта, если такой объект существует и уникален, иначе возвращает nullptr;

- Параметры:
  - Строковой параметр s_name - наименование текущего объекта;
- Тип возвращаемого значения: объект класса cl_base;
- Модификатор доступа: public;

- Метод find_object_from_current
  - Функционал: поиск объекта с заданным наименованием в дереве объектов, начиная с текущего объекта.
  - Параметры:
    - Строковой параметр s_name - наименование текущего объекта;
  - Тип возвращаемого значения: объект класса cl_base;
  - Модификатор доступа: public;
- Метод find_object_from_current2:
  - Функционал: поиск объекта с заданным наименованием в дереве объектов, начиная с текущего объекта.
  - Параметры:
    - Строковой параметр s_name - наименование текущего объекта;
  - Тип возвращаемого значения: объект класса cl_base;
  - Модификатор доступа: public;
- Метод find_object_from_root:
  - Функционал: поиск объекта с заданным

наименованием в дереве объектов, начиная с текущего объекта.

- Параметры:
  - Строковой параметр s_name - наименование текущего объекта;
- Тип возвращаемого значения: объект класса cl_base;
- Модификатор доступа: public;

- Метод set_state:
  - Функционал: изменения состояния объекта;
  - Параметры:
    - Целочисленный параметр new_state - установка состояния объекта;
  - Тип возвращаемого значения: нет;
  - Модификатор доступа: public;
- Метод can_be_ready:
  - Функционал: поверяет, можно ли объект готовить к использованию на основе его текущего состояния;
  - Параметры: нет;
  - Тип возвращаемого значения: булевое;
  - Модификатор доступа: public;

- Класс cl_application наследуется от класса cl_base:
  - Поля: наследуемые от класса cl_base;
  - Методы:
    - Метод build_tree_objects:
      - Функционал: отвечает за создание иерархии объектов и установку их сязей;
      - Параметры: нет;

        - Тип возвращаемого значения: нет;
        - Модификатор доступа: public;
    - Метод set_object_ready:
        - Функционал: устанавливает объект p_object в готовое состояние с помощью метода set_state;
        - Параметры:
            - Указатель p_object на объект класса cl_base;
            - Целочисленный праметр state_number - значение, которое будет установлено в качестве состояния объекта.
        - Тип возвращаемого значения: нет;
        - Модификатор доступа: public;
- Класс cl_1 наследуется от класса cl_base:
    - Поля: наследуемые от класса cl_base;
    - Методы:
        - Параметризированный конструктор cl_1:
            - Функционал: параметризированный констркуктор который в качестве параметров принимает указатель на головной объект в дереве иерархии и возмижно наименование текущего объекта;
            - Параметры:
                - Указатель head_objectt на объект класса cl_base - отвечает за доступа к головному объекту текущего объекта;
                - Строковый параметр s_name - возможное наиенование текущего объекта;
            - Тип возвращаемого значения: объект класса cl_base;

        - Модификатор доступа: public;
- Класс cl_2 наследуется от класса cl_base:
    - Поля: наследуемые от класса cl_base;
    - Методы:
        - Параметризированный конструктор cl_2:
            - Функционал: параметризированный констркуктор который в качестве параметров принимает указатель на головной объект в дереве иерархии и возмижно наименование текущего объекта;
            - Параметры:
                - Указатель head_objectt на объект класса cl_base - отвечает за доступа к головному объекту текущего объекта;
                - Строковый параметр s_name - возможное наиенование текущего объекта;
            - Тип возвращаемого значения: объект класса cl_base;
            - Модификатор доступа: public;
- Класс cl_3 наследуется от класса cl_base:
    - Поля: наследуемые от класса cl_base;
    - Методы:
        - Параметризированный конструктор cl_3:
            - Функционал: параметризированный констркуктор который в качестве параметров принимает указатель на головной объект в дереве иерархии и возмижно наименование текущего объекта;
            - Параметры:
                - Указатель head_objectt на объект класса cl_base -

отвечает за доступа к головному объекту текущего объекта;

- Строковый параметр s_name - возможное наиенование текущего объекта;

- Тип возвращаемого значения: объект класса cl_base;
- Модификатор доступа: public;

- Класс cl_4 наследуется от класса cl_base:
  - Поля: наследуемые от класса cl_base;
  - Методы:
    - Параметризированный конструктор cl_4:
      - Функционал: параметризированный констркуктор который в качестве параметров принимает указатель на головной объект в дереве иерархии и возмижно наименование текущего объекта;
      - Параметры:
        - Указатель head_objectt на объект класса cl_base - отвечает за доступа к головному объекту текущего объекта;
        - Строковый параметр s_name - возможное наиенование текущего объекта;
      - Тип возвращаемого значения: объект класса cl_base;
      - Модификатор доступа: public;
- Класс cl_5 наследуется от класса cl_base:
  - Поля: наследуемые от класса cl_base;
  - Методы:
    - Параметризированный конструктор cl_5:
      - Функционал: параметризированный констркуктор

который в качестве параметров принимает указатель на головной объект в дереве иерархии и возмижно наименование текущего объекта;

- Параметры:
  - Указатель head_objectt на объект класса cl_base - отвечает за доступа к головному объекту текущего объекта;
  - Строковый параметр s_name - возможное наиенование текущего объекта;
- Тип возвращаемого значения: объект класса cl_base;
- Модификатор доступа: public;

- Класс cl_6 наследуется от класса cl_base:
  - Поля: наследуемые от класса cl_base;
  - Методы:
    - Параметризированный конструктор cl_6:
      - Функционал: параметризированный констркуктор который в качестве параметров принимает указатель на головной объект в дереве иерархии и возмижно наименование текущего объекта;
      - Параметры:
        - Указатель head_objectt на объект класса cl_base - отвечает за доступа к головному объекту текущего объекта;
        - Строковый параметр s_name - возможное наиенование текущего объекта;
      - Тип возвращаемого значения: объект класса cl_base;
      - Модификатор доступа: public;

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 print_tree класса cl_base

Функционал: Вывод на экран состояние объекта и его подчиненных объектов в виде дерева.

Параметры: Целочисленный параметр level - определяющий отступ вывода объектов.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация переменной i = 0	2
2	i < level	Вывод отступа, размеров в 4 пробела с помощью объекта cout	3
			4
3		i++	2
4		Вывод имени текущего объекта	5
5	Вектор sub_objects текущего объекта еще содержит нерассмотренные элементы	Рекурсивный вызов метода print_tree с увеличенным на 1 значением параметра level	5
			∅

3.2 Алгоритм метода count класса cl_base

Функционал: Возвращает количество объектов с заданнфм наименованием в иерархии объектов.

Параметры: Строковой параметр s_name - наименование текущего объекта.

Возвращаемое значение: Возврат целочисленного значения counter.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода count класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация переменной counter = 0	2
2	Провекра на соответствие значения имени текущего объекта s_name	Увеличиваем счётчик counter на единицу	3
			3
3	Итерация по всем подчиненным объектам sub_objects	Увеличение счетчика counter на значение, которое возвращает метод count для каждого подчиненного объекта p_sub_object	3
			4
4		Возврат значения counter	∅

3.3 Алгоритм метода search_object класса cl_base

Функционал: Возвращает указатель на объект с заданным наименованием в иерархии объектов.

Параметры: Строковой параметр s_name - наименование текущего объекта.

Возвращаемое значение: Указатель на объект cl_base.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода search_object класса cl_base

№	Предикат	Действия	№ перехода
1	Количество объектов с заданным именем не равно 1	Возврат пустого множества	∅
			2
2	Имя текущего объекта совпадает с заданным именем	Возврат имени текущего объекта	∅
			3
3	Перебраны все подобъекта текущего объекта		6
			4
4		Вызов метода search_object для текущего под объекта	5
5	Текущий подобъект имеет указатель на объект	Вернуть этот указатель	∅
			3
6		Возврат пустого множества	∅

3.4 Алгоритм метода find_object_from_current класса cl_base

Функционал: Поиск объекта с заданным наименованием в дереве объектов.

Параметры: Строковой параметр s_name - наименование текущего объекта.

Возвращаемое значение: Указатель на объект cl_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода find_object_from_current класса cl_base

№	Предикат	Действия	№ перехода
1	Количество объектов с заданным именем не равно 1	Возврат пустого множества	∅

№	Предикат	Действия	№ перехода
			2
2		Вызов метода search_object и вернуть его значение	∅

3.5 Алгоритм метода find_object_from_current2 класса cl_base

Функционал: Поиск объекта с заданным наименованием в дереве объектов.

Параметры: Строковой параметр s_name - наименование текущего объекта.

Возвращаемое значение: Указатель на объект cl_base.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода find_object_from_current2 класса cl_base

№	Предикат	Действия	№ перехода
1		Создание очереди q	2
2		Инициализация указателя p_found значением nullptr	3
3		Добавление текущего объекта this в очередь q	4
4	Очередь q не пуста		5
			10
5	Передняя часть очереди q равна заданному имени s_name		6
			7
6	Указатель p_found имеет значение nullptr	Присваиваем арес объекта в передней части очереди	4
		Возврат nullptr	4
7	Перебраны все подобъекта текущего объекта		9
			8
8		Добавляем подобъект текущего объекта в конец	7

№	Предикат	Действия	№ перехода
		очереди q	
9		Удаление объекта из передней части очереди q	4
10		Возврат указателя p_found	∅

3.6 Алгоритм метода find_object_from_root класса cl_base

Функционал: Поиск объекта с заданным наименованием в дереве объектов.

Параметры: Строковой параметр s_name - наименование текущего объекта.

Возвращаемое значение: Указатель на объект cl_base.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода find_object_from_root класса cl_base

№	Предикат	Действия	№ перехода
1	Текущий объект не содержит объектов с именем s_name	Вернуть nullptr	∅
			2
2		Создать указатель p_current и установить его на текущий объект	3
3	У текущего объекта есть головной объект	Устанавить p_current на головной объект	3
			4
4		Вернуть результат метода search_object у объекта p_current с аргументом s_name	∅

3.7 Алгоритм метода set_state класса cl_base

Функционал: Изменения состояния объекта.

Параметры: Строковой параметр s_name - наименование текущего объекта.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода set_state класса cl_base

№	Предикат	Действия	№ перехода
1		Присвоить нового значения new_state переменной state	∅

3.8 Алгоритм метода can_be_ready класса cl_base

Функционал: Проверка, можно ли объект готовить к использованию на основе его текущего состояния;.

Параметры: Нет.

Возвращаемое значение: Логическое значение.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода can_be_ready класса cl_base

№	Предикат	Действия	№ перехода
1		Создать указатель root на объект, равный this	2
2	У корневого объекта есть родитель		3
			4
3	У родителя корневого объекта установлено состояние	Указатель root переносится на родительский объект	2
		Вернуть false	∅
4		Вернуть true	∅

3.9 Алгоритм метода build_tree_objects класса cl_application

Функционал: Отвечает за создание иерархии объектов и установку их сязей.

Параметры: Нет.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Инициализация строковых переменных s_head_name,s_sub_name	2
2		Инициализация указателя p_head на текущий объект this	3
3		Инициализация указателя p_sub = nullptr	4
4		Инициализация целочисленных переменных i_class, i _state	5
5		Ввод значения s_head_name	6
6		Вызвать метод set_name текущего объекта с параметром s_head_name	7
7	Истина		8
			18
8		Ввод значения s_head_name	9
9	s_head_name равен "endtree"	Выйти из цикла	18
			10
10		Ввод значений s_sub_name и i_class	11
11		Найти объект в дереве с именем s_head_name с помощью метода find_object_from_room и присвоить его переменной p_head	12
12	i_class == 1	Присвоить p_head созданный объект cl_1 с параметрами p_head, s_sub_name	7
			13
13	i_class == 2	Присвоить p_head созданный объект cl_2 с параметрами p_head, s_sub_name	7

№	Предикат	Действия	№ перехода
			14
14	i_class == 3	Присвоить p_head созданный объект cl_3 с параметрами p_head, s_sub_name	7
			15
15	i_class == 4	Присвоить p_head созданный объект cl_4 с параметрами p_head, s_sub_name	7
			16
16	i_class == 5	Присвоить p_head созданный объект cl_5 с параметрами p_head, s_sub_name	7
			17
17	i_class == 6	Присвоить p_head созданный объект cl_6 с параметрами p_head, s_sub_name	7
			7
18	Ввод значения s_head_name	Ввод значения i_state	19
			21
19		Найти объект в дереве с именем s_head_name с помощью метода find_object_froom_root и присвоить его переменной p_head	20
20	Объект и все его потомки готовы	Вывзать метод set_readt для объекта с параметром i_state	18
			18
21			∅

3.10 Алгоритм метода set_object_ready класса cl_application

Функционал: Устанавливает объект p_object в готовое состояние с помощью метода set_state.

Параметры: Указатель p_object на объект класса cl_base, целочисленный праметр state_number .

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода set_object_ready класса cl_application

№	Предикат	Действия	№ перехода
1	Переданный указатель на объект является нулевым		∅
			2
2		Вывзов метода set_state для переданного объекта с аргументом state_number	∅

3.11 Алгоритм конструктора класса cl_1

Функционал: Создание объекта элемента дерева.

Параметры: Указатель на объекта класса cl_base head_object, s_name - имя объекта.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_1

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора класса cl_1 и передача в него в качестве параметра значения параметра head_objects и s_name	∅

3.12 Алгоритм конструктора класса cl_2

Функционал: Создание объекта элемента дерева.

Параметры: Указатель на объекта класса cl_base head_object, s_name - имя объекта.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора класса cl_2 и передача в него в качестве параметра значения параметра head_objects и s_name	∅

3.13 Алгоритм конструктора класса cl_3

Функционал: Создание объекта элемента дерева.

Параметры: Указатель на объекта класса cl_base head_object, s_name - имя объекта.

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора класса cl_3 и передача в него в качестве параметра значения параметра head_objects и s_name	∅

3.14 Алгоритм конструктора класса cl_4

Функционал: Создание объекта элемента дерева.

Параметры: Указатель на объекта класса cl_base head_object, s_name - имя объекта.

Алгоритм конструктора представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора класса cl_4 и передача в него в качестве параметра значения параметра head_objects и s_name	∅

3.15 Алгоритм конструктора класса cl_5

Функционал: Создание объекта элемента дерева.

Параметры: Указатель на объекта класса cl_base head_object, s_name - имя объекта.

Алгоритм конструктора представлен в таблице 15.

Таблица 15 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора класса cl_5 и передача в него в качестве параметра значения параметра head_objects и s_name	∅

3.16 Алгоритм конструктора класса cl_6

Функционал: Создание объекта элемента дерева.

Параметры: Указатель на объекта класса cl_base head_object, s_name - имя объекта.

Алгоритм конструктора представлен в таблице 16.

Таблица 16 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора класса cl_6 и передача в него в качестве параметра значения параметра head_objects и s_name	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-15.

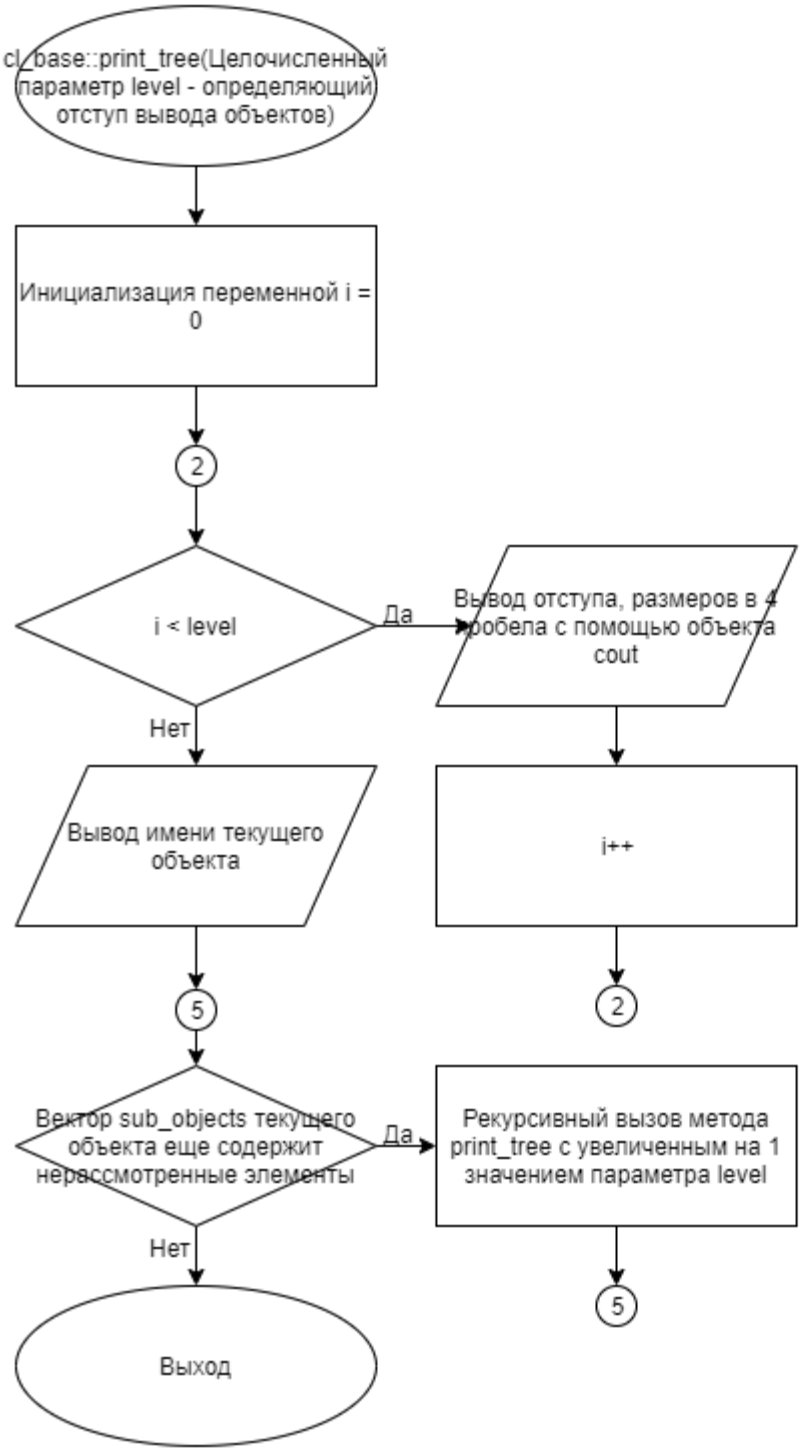

Рисунок 1 – Блок-схема алгоритма

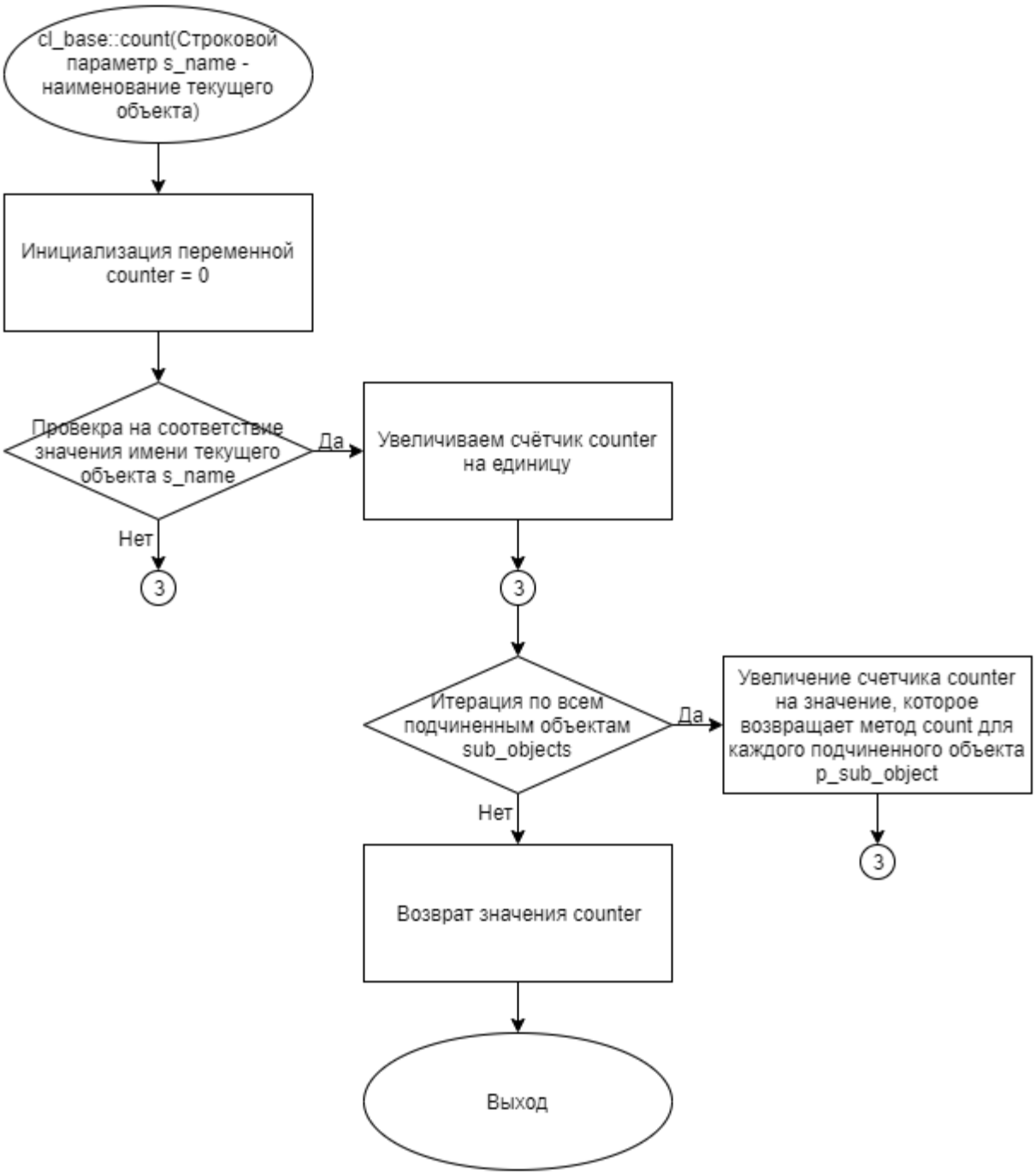

Рисунок 2 – Блок-схема алгоритма

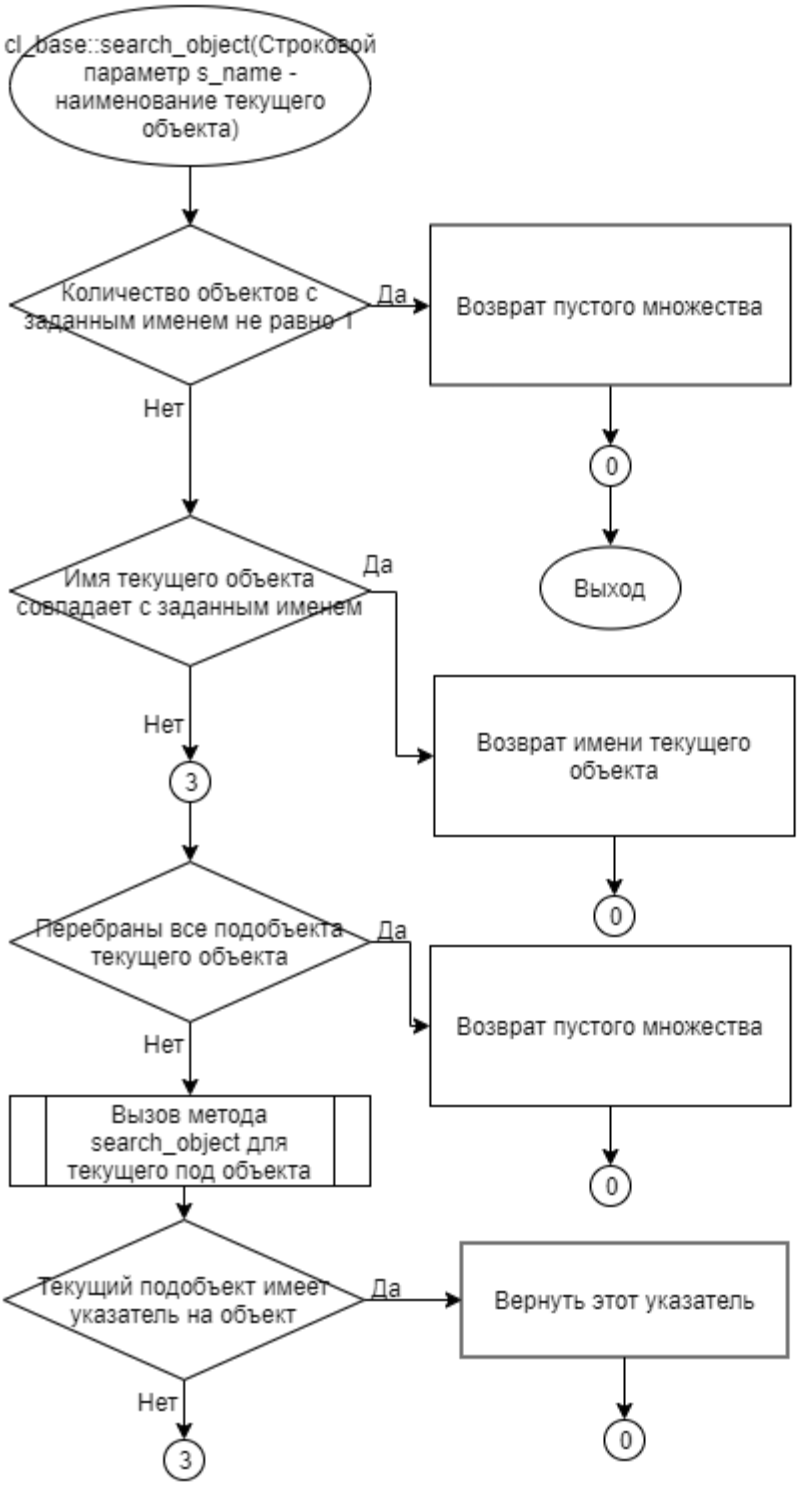

Рисунок 3 – Блок-схема алгоритма

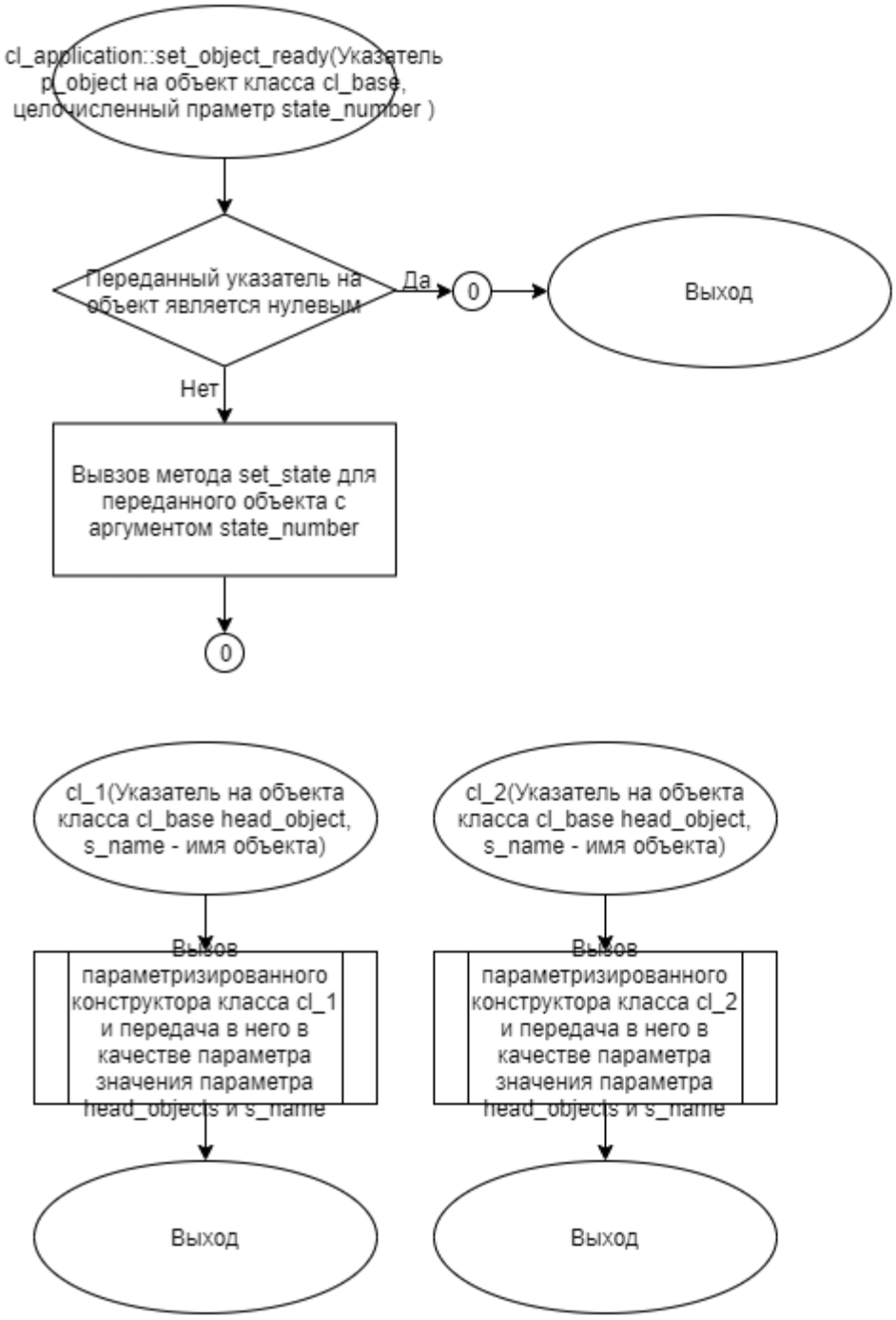

Рисунок 4 – Блок-схема алгоритма

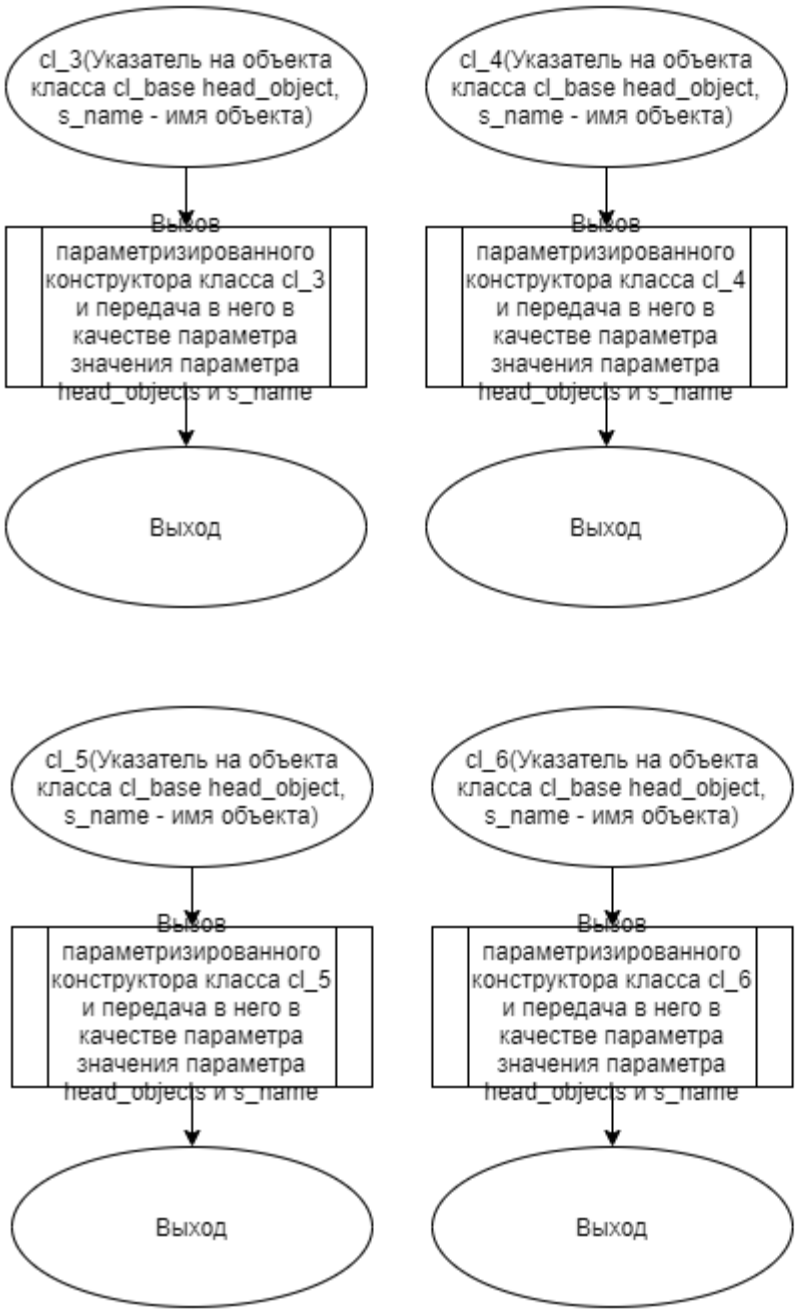

Рисунок 5 – Блок-схема алгоритма

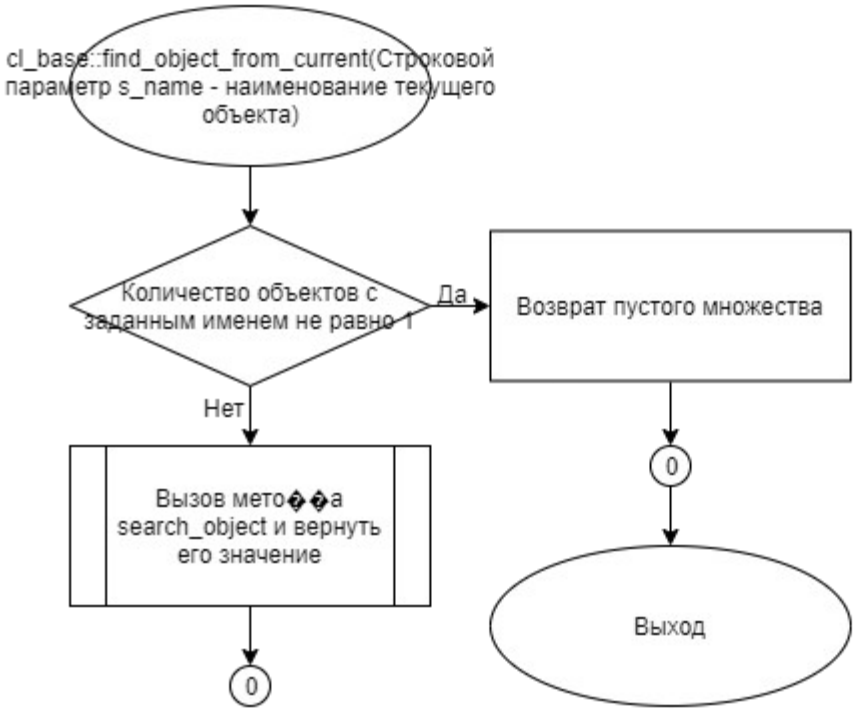

Рисунок 6 – Блок-схема алгоритма

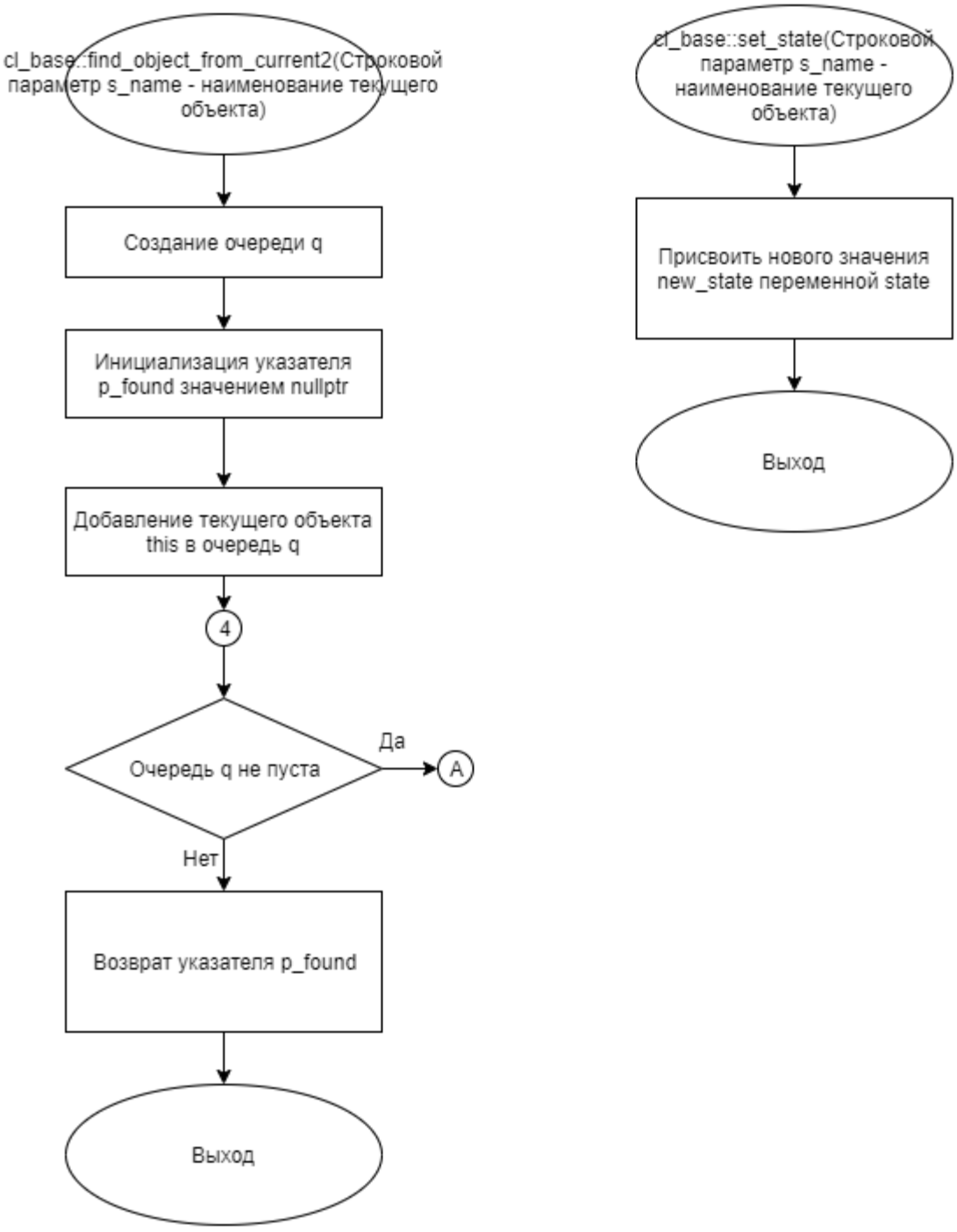

Рисунок 7 – Блок-схема алгоритма

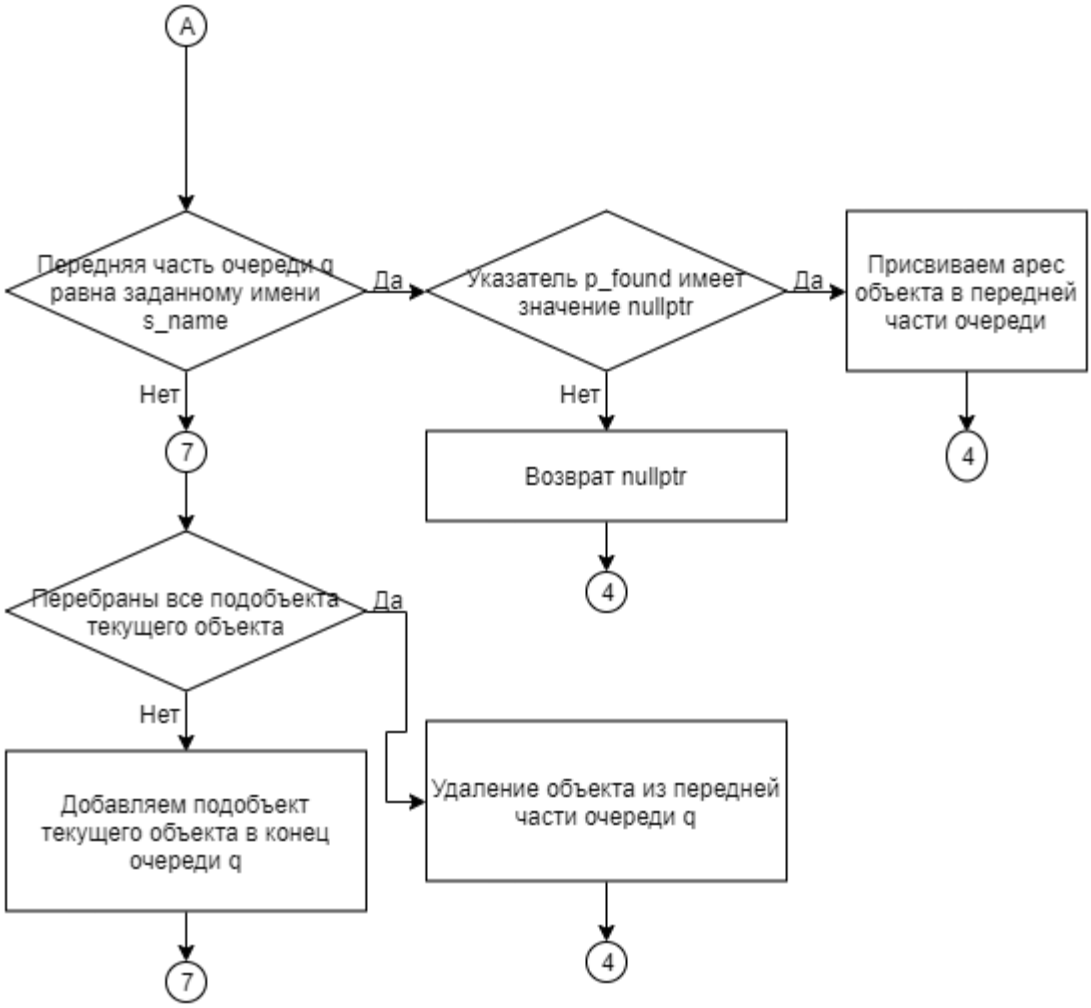

Рисунок 8 – Блок-схема алгоритма

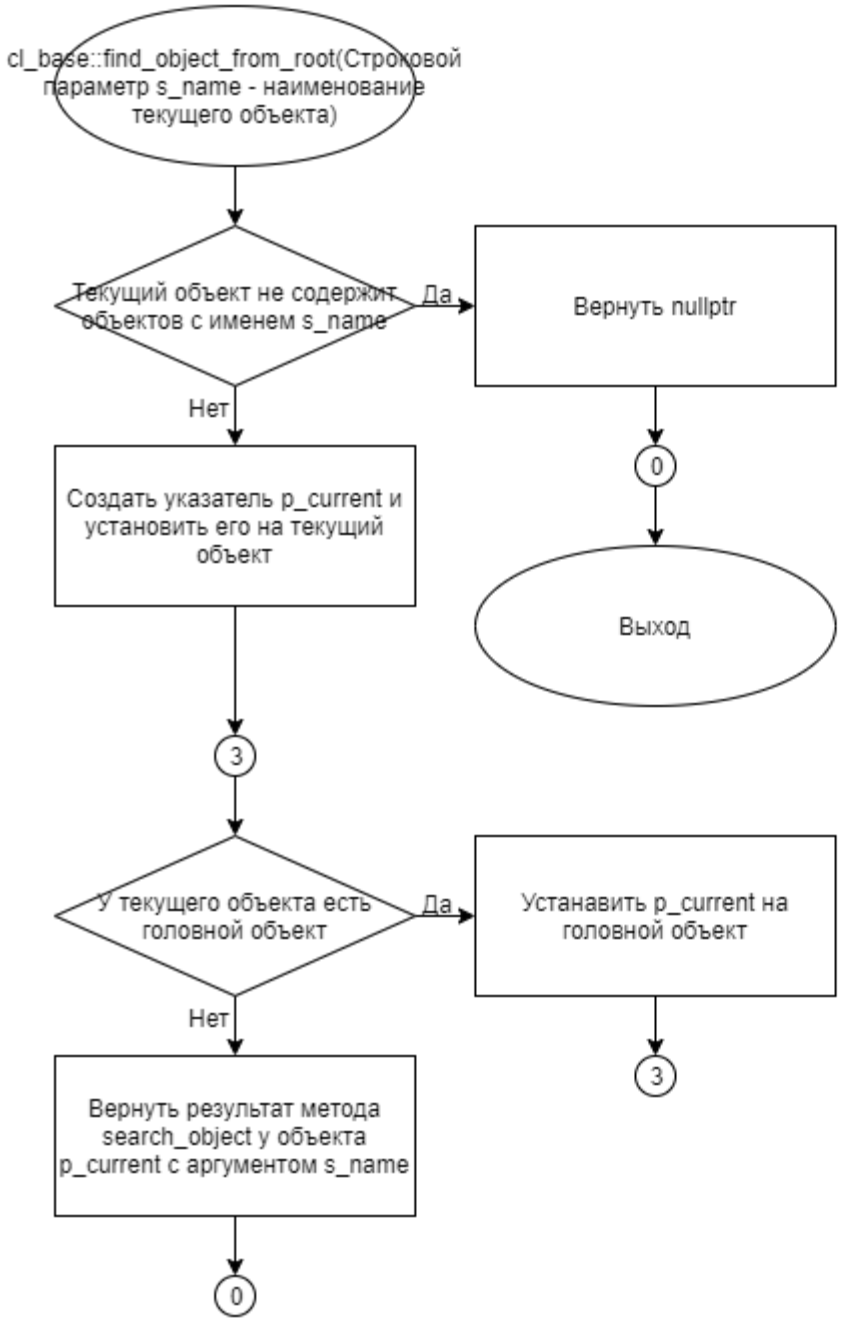

Рисунок 9 – Блок-схема алгоритма

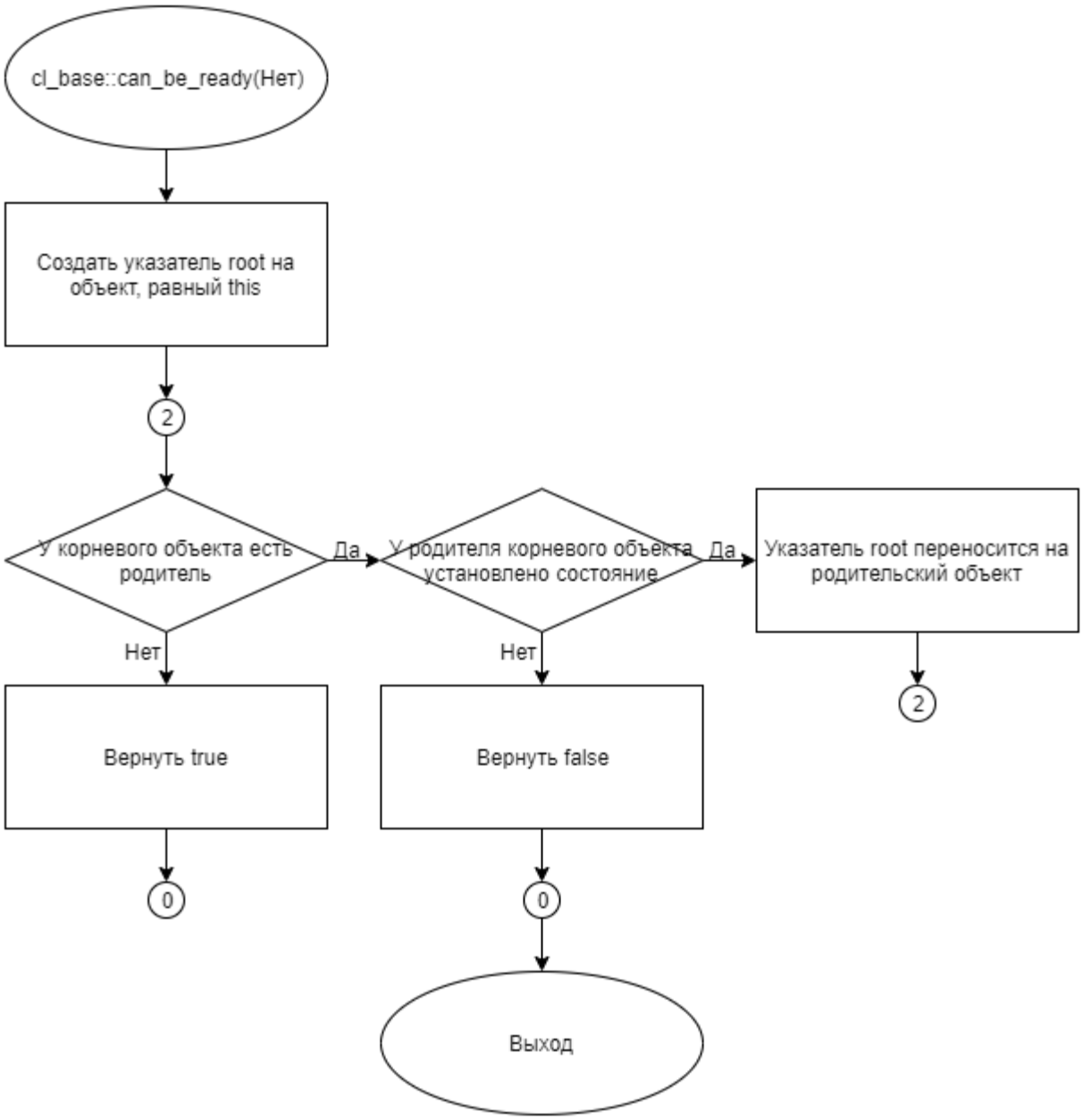

Рисунок 10 – Блок-схема алгоритма

Рисунок 11 – Блок-схема алгоритма

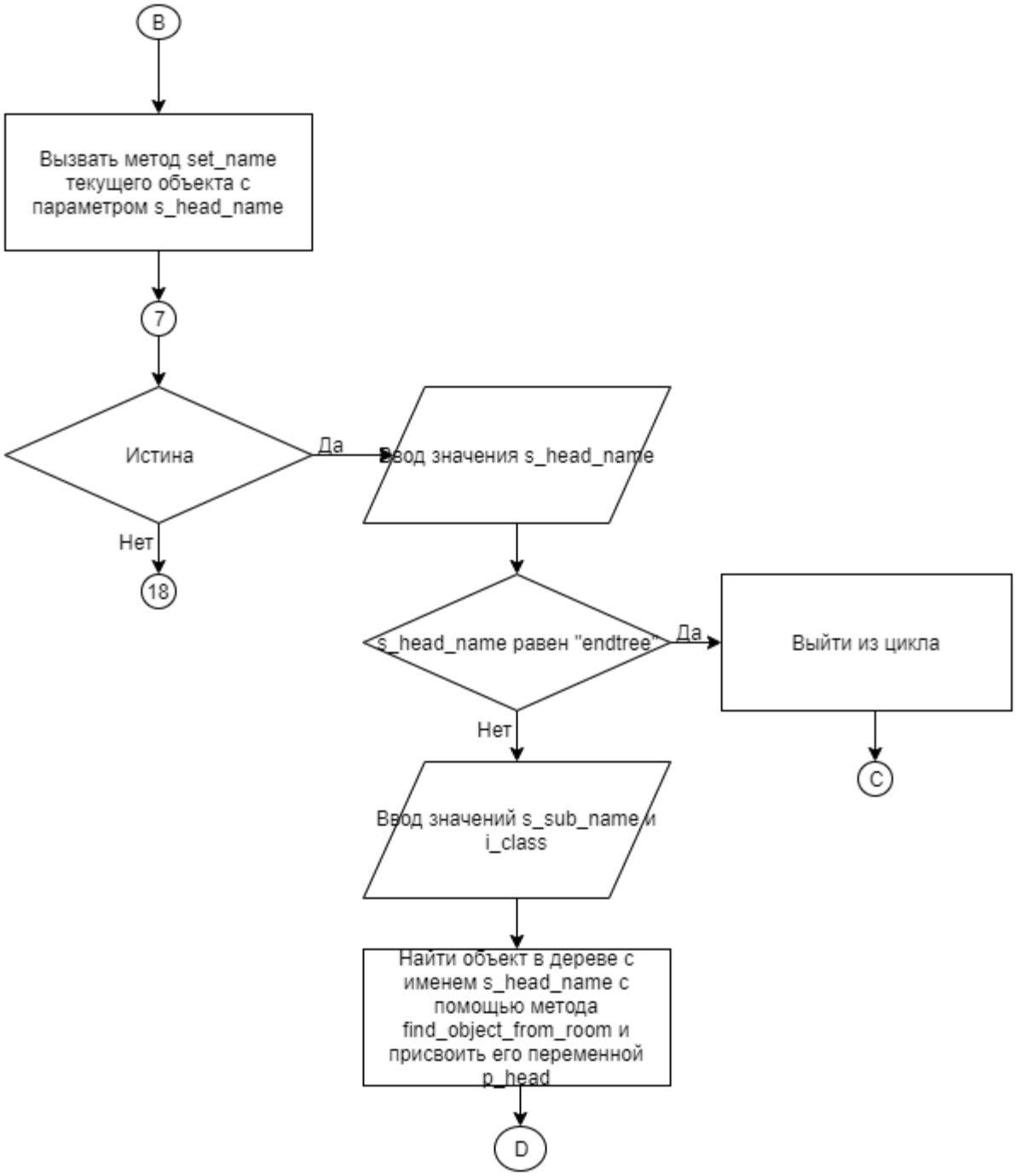

Рисунок 12 – Блок-схема алгоритма

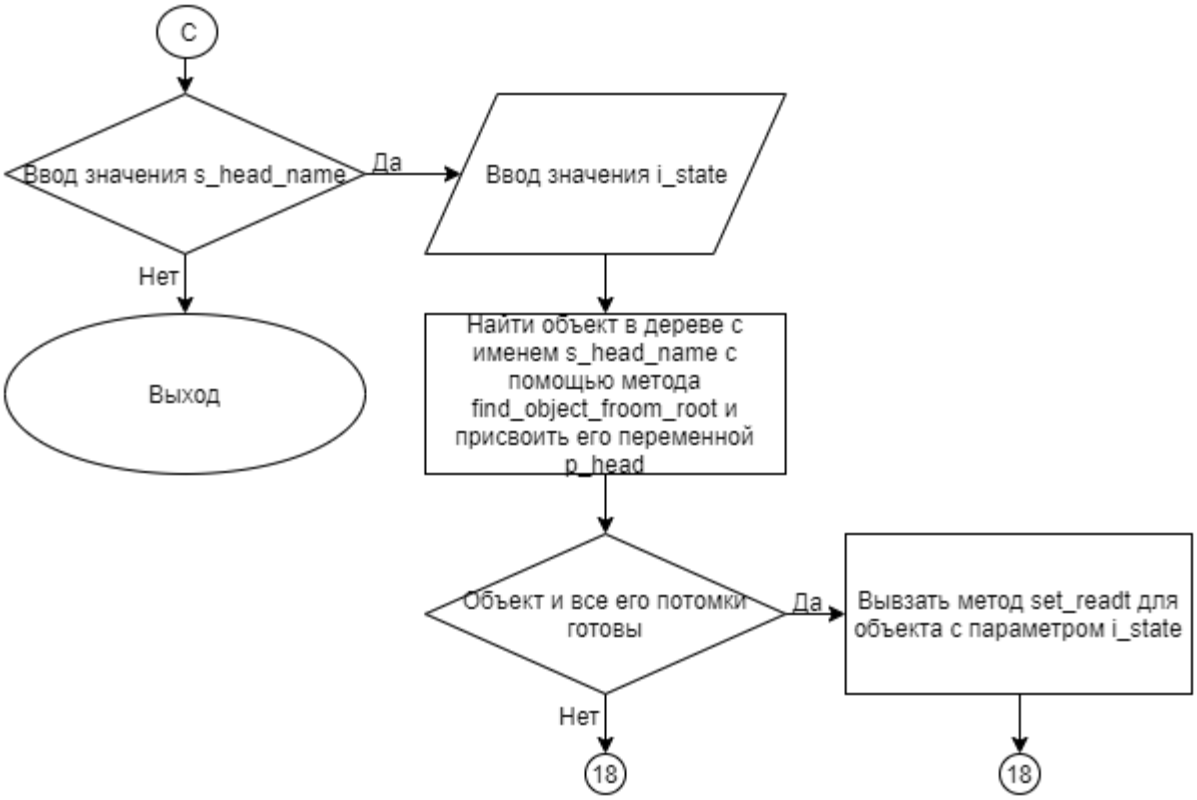

Рисунок 13 – Блок-схема алгоритма

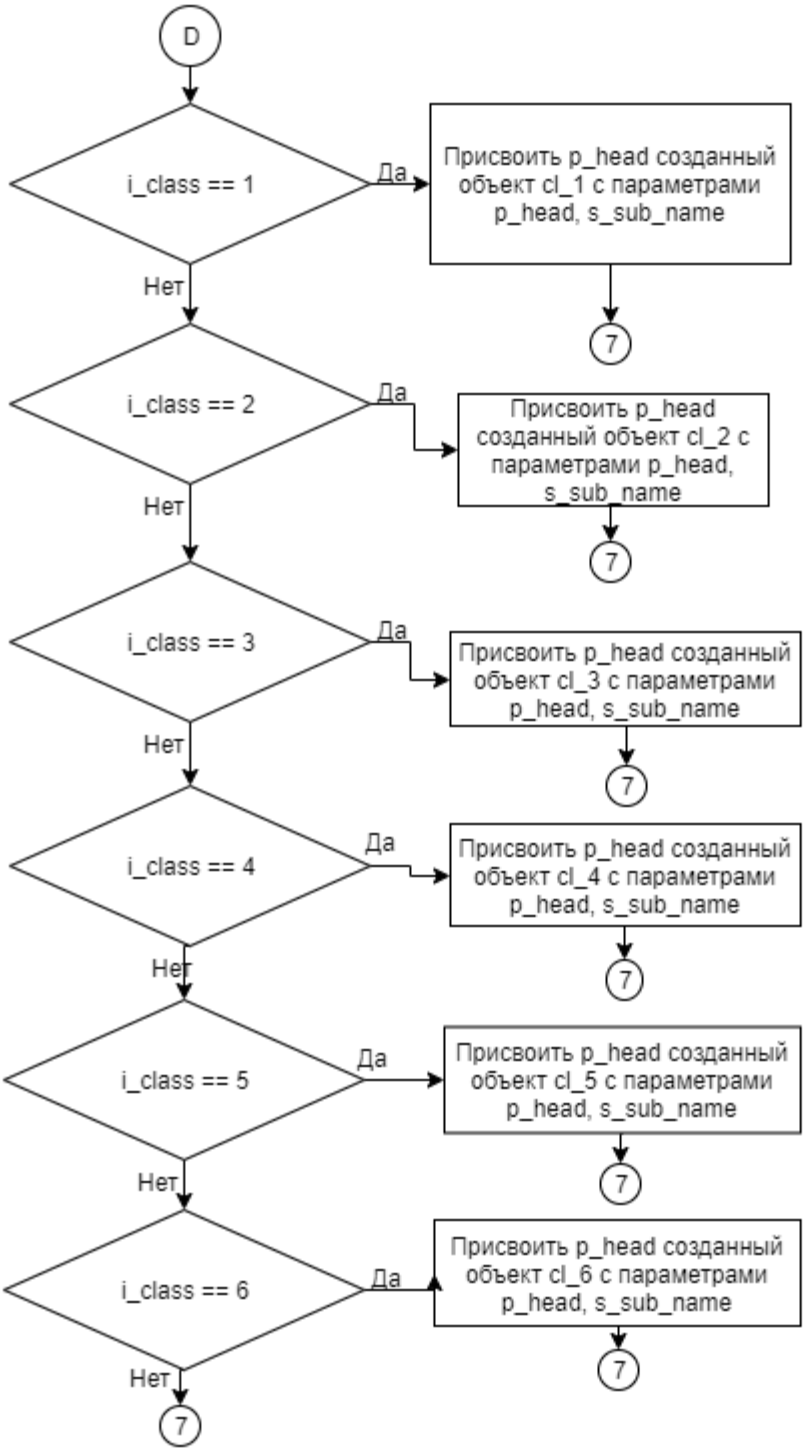

Рисунок 14 – Блок-схема алгоритма

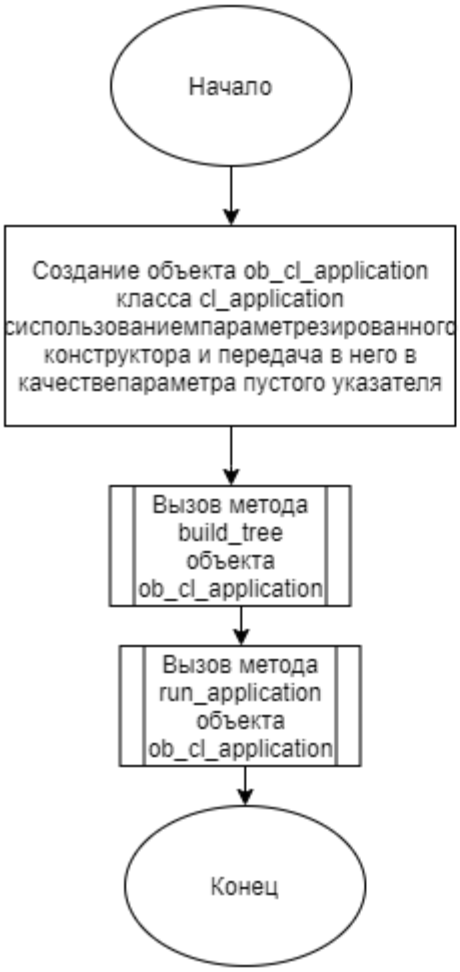

Рисунок 15 – Блок-схема алгоритма

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*  head_object,  string  s_name)  :  cl_base(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* 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*  head_object,  string  s_name)  :  cl_base(head_object,
s_name) {};
```

5.4 Файл cl_2.h

Листинг 4 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H

#include "cl_base.h"

class cl_2 : public cl_base {
public:
   cl_2(cl_base* head_object, string s_name);
};

#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base*  head_object,  string  s_name)  :  cl_base(head_object,
s_name) {};
```

5.6 Файл cl_3.h

Листинг 6 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"
class cl_3 : public cl_base{
public:
   cl_3(cl_base* head_object, string s_name);
};

#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base*  head_object,  string  s_name)  :  cl_base(head_object,
s_name) {};
```

5.8 Файл cl_4.h

Листинг 8 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include "cl_base.h"
class cl_4 : public cl_base{
public:
   cl_4(cl_base* head_object, string s_name);
};

#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base*  head_object,  string  s_name)  :  cl_base(head_object,
s_name) {};
```

5.10 Файл cl_5.h

Листинг 10 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
```

```
#include "cl_base.h"
class cl_5 : public cl_base{
public:
   cl_5(cl_base* head_object, string s_name);
};

#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base*  head_object,  string  s_name)  :  cl_base(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* head_object, string s_name);
};

#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.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"
// Конструктор класса cl_application, принимает указатель на корневой объект
и устанавливает его как родительский.
cl_application::cl_application(cl_base*  head_object)  :  cl_base(head_object)
{

};

// Метод для построения дерева объектов. Считывает с консоли имя объекта,
затем имя подчиненного объекта,
// его тип и создает новый объект нужного типа. Пока не встретится ключевое
слово "endtree".
// Затем считывает состояние объектов и устанавливает соответствующий флаг
готовности.
void cl_application::build_tree_objects(){
   string s_head_name, s_sub_name;
   cl_base* p_head = this;
   cl_base* p_sub = nullptr;
   int i_class, i_state;
   cin >> s_head_name;
   this->set_name(s_head_name);

   while (true) {
      cin >> s_head_name;
      if (s_head_name == "endtree")
         break;
      cin >> s_sub_name >> i_class;
      p_head = find_object_from_root(s_head_name);
      switch(i_class){
         case 1:
            p_head = new cl_1(p_head,s_sub_name);
            break;
         case 2:
            p_head = new cl_2(p_head,s_sub_name);
            break;
         case 3:
            p_head = new cl_3(p_head,s_sub_name);
            break;
         case 4:
            p_head = new cl_4(p_head,s_sub_name);
            break;
         case 5:
            p_head = new cl_5(p_head,s_sub_name);
            break;
         case 6:
            p_head = new cl_6(p_head,s_sub_name);
            break;
         }
   }
   while (cin >> s_head_name)
   {
      cin >> i_state;
      p_head = this->find_object_from_root(s_head_name);
```

```
            if(p_head->can_be_ready()){
               p_head->set_ready(i_state);
            }
         }
      }

      // Метод для установки состояния объекта. Принимает указатель на объект и
      номер состояния.
      void cl_application::set_object_ready(cl_base* p_object, int state_number) {
         if (!p_object) {
            return;
         }
         p_object->set_state(state_number);
      }

      // Метод для запуска приложения, выводит дерево объектов и их готовность в
      консоль.
      int cl_application::exec_app() {
         int level = 0;
         cout << "Object tree" << endl;
         print_tree(level);
         cout << "The tree of objects and their readiness";
         print_status(level);
         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"
      //Класс cl_application, главный класс приложения, наследуется от базового
      класса cl_base.
      class cl_application : public cl_base {
      public:
         // Конструктор класса, принимает указатель на корневой объект.
         cl_application(cl_base* head_object);
         // Конструктор класса, принимает указатель на корневой объект.
         void build_tree_objects();
         // Метод для запуска приложения, выводит дерево объектов и их готовность в
      консоль.
         int exec_app();
         // Метод для установки состояния объекта. Принимает указатель на объект и
      номер состояния.
         void set_object_ready(cl_base* p_object, int state_number);
```

```
};

#endif
```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
#include "cl_base.h"

// Конструктор класса, который принимает указатель на объект-родитель и имя создаваемого объекта
cl_base::cl_base(cl_base* head_object, string s_name) {
   // Присваиваем имя объекту
   this->s_name = s_name;
   // Указываем родительский объект
   this->p_head_object = head_object;

   // Если родительский объект существует, то добавляем текущий объект как его подобъект
   if (p_head_object != nullptr)
      p_head_object->sub_objects.push_back(this);
}


// Деструктор класса
cl_base::~cl_base() {
   // Рекурсивно удаляем все подобъекты данного объекта
   for (int i = 0; i < this->sub_objects.size(); i++) {
      delete sub_objects[i];
   }
}

// Метод для изменения имени объекта
bool cl_base::set_name(string s_new_name) {
   // Если объект имеет родителя, то проверяем, что другой объект с таким же именем не является подобъектом данного родителя
   if (p_head_object != nullptr) {
      for (int i = 0; i < p_head_object->sub_objects.size(); i++) {
         if (p_head_object->sub_objects[i]->get_name() == get_name())
            return false;
      }
   }
   // Иначе меняем имя объекта
   this->s_name = s_new_name;
   return true;
}

// Метод для получения имени объекта
string cl_base::get_name() {
```

```
   return this->s_name;
}

// Метод для проверки, можно ли данный объект пометить как готовый
bool cl_base::can_be_ready(){
   // Начинаем с текущего объекта
   cl_base* root = this;
   // Пока у объекта есть родительский объект, проверяем, что он готов, иначе
возвращаем false
   while(root->p_head_object){
      if(root->p_head_object->state){
         root = root->p_head_object;
      }
      else{
         return false;
      }
   }
   // Если дошли до корневого объекта и все объекты на пути были готовы, то
возвращаем true
   return true;
}

// Метод для получения указателя на родительский объект
cl_base* cl_base::get_head_object() {
   return this->p_head_object;
}

// Метод для получения указателя на подобъект с заданным именем
cl_base* cl_base::get_sub_object(string s_name) {
   for (int i = 0; i < this->sub_objects.size(); i++) {
      if (sub_objects[i]->get_name() == s_name)
         return this->sub_objects[i];
   }
   return nullptr;
}
// Метод для печати дерева объектов, начиная с данного объекта
void cl_base::print_tree(int level) {
   // Выводим отступы в соответствии с уровнем вложенности и имя объекта
   for(int i = 0; i< level;i++){
      cout << "    ";
   }
   cout << s_name << endl;
   // Рекурсивно вызываем метод для каждого подобъекта
   for(auto sub_object : sub_objects){
      sub_object->print_tree(level+1);
   }
}

void cl_base::print_status(int level){
   cout <<endl;
   for(int i = 0; i < level;i++) cout << "    ";
   if(state) cout <<  s_name << " is ready" ;
   else cout << s_name <<  " is not ready" ;

   for(auto sub_object : sub_objects){
```

```
            sub_object->print_status(level+1);
        }
    }
    // Функция подсчета количества объектов с заданным именем в дереве объектов
    int cl_base::count(string s_name)
    {
        int counter = 0;
        // Если имя текущего объекта соответствует заданному, увеличиваем счетчик
    на 1
        if (this->get_name() == s_name)
            counter++;
        //  Рекурсивный обход всех подобъектов текущего объекта и подсчет
    количества объектов с заданным именем
        for (auto p_sub_object : sub_objects)
            counter += p_sub_object->count(s_name);

        return counter;
    }


    // Функция поиска объекта с заданным именем в дереве объектов
    cl_base* cl_base::search_object(string s_name)
    {
        // Если количество объектов с заданным именем не равно 1, значит такой
    объект не найден или найдено более одного
        if (this->count(s_name) != 1)
            return nullptr;
        // Если имя текущего объекта соответствует заданному, возвращаем указатель
    на этот объект
        if (this->get_name() == s_name)
            return this;

        // Рекурсивный поиск объекта с заданным именем в подобъектах текущего
    объекта
        for (auto p_sub_object : sub_objects)
        {
            cl_base* p_found = p_sub_object->search_object(s_name);
            if (p_found != nullptr)
            {
                return p_found;
            }
        }

        return nullptr;
    }


    // Функция поиска объекта с заданным именем начиная с текущего объекта
    cl_base* cl_base::find_object_from_current(string s_name)
    {

        // Если количество объектов с заданным именем не равно 1, значит такой
    объект не найден или найдено более одного
        if (this->count(s_name) != 1)
        {
```

```
            return nullptr;
        }
        // Рекурсивный поиск объекта с заданным именем начиная с текущего объекта
        return search_object(s_name);
    }

    // Функция поиска объекта с заданным именем в ширину начиная с текущего
    объекта
    cl_base* cl_base::find_object_from_current2(string s_name)
    {
        queue<cl_base*> q;
        cl_base* p_found = nullptr;

        q.push(this);

        while (!q.empty())
        {
            // Проверяем объект в начале очереди на соответствие заданному имени
            if (q.front()->get_name() == s_name)
            {
                if  (p_found  ==  nullptr)//  Если  это  первый  найденный  объект,
    запоминаем его
                {
                    p_found = q.front();
                }
                else // Если уже был найден ранее другой объект с таким же именем,
    возвращаем nullptr
                {
                    return nullptr;
                }
            }
            // Добавляем все подобъекты текущего объекта в конец очереди
            for (auto p_sub_object : sub_objects)
                q.push(p_sub_object);
            // Удаляем текущий объект из начала очереди
            q.pop();
        }

        return p_found;
    }
    // Удаляем текущий объект из начала очереди
    void cl_base::set_state(int new_state) {
        state = new_state;
    }

    // Функция поиска объекта с заданным именем начиная с корневого объекта
    cl_base* cl_base::find_object_from_root(string s_name)
    {
        // Если количество объектов с заданным именем не равно 1, значит такой
    объект не найден или найдено более одного
        if (this->count(s_name) != 1)
        {
            return nullptr;
        }
        cl_base* p_current = this;
        while (p_current->get_head_object() != nullptr) // Идем вверх по дереву
```

```
объектов до корневого объекта
   {
      p_current = p_current->get_head_object();
   }
   return p_current->search_object(s_name);
}
// Метод set_ready устанавливает флаг готовности объекта.
//  Если  он  равен  нулю,  то  также  вызывает  метод  set_ready  для  всех
подчиненных объектов.
void cl_base::set_ready(int state){
   this->state = state;
   if(!state){
      for(auto& el: sub_objects ){
         el->set_ready(state);
      }
   }
}
```

5.16 Файл cl_base.h

Листинг 16 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H

#include <vector>
#include <string>
#include <iostream>
#include <queue>
using namespace std;

// Класс cl_base, базовый класс для всех объектов в приложении.
class cl_base {
private:
   string s_name; // Имя объекта.
   cl_base* p_head_object; // Указатель на родительский объект.
   int state; // Указатель на родительский объект.
   vector  <cl_base*>  sub_objects;  //  Вектор  указателей  на  подчиненные
объекты.
public:
   // Конструктор класса, принимает указатель на родительский объект и имя
объекта.
   cl_base(cl_base* head_object, string s_name = "Base object");
   // Деструктор класса.
   ~cl_base();
   // Метод для установки имени объекта. Принимает новое имя и возвращает
true, если удалось установить имя.
   bool set_name(string new_name);
   // Метод для получения имени объекта.
   string get_name();
```

```
    // Метод для получения указателя на родительский объект.
    cl_base* get_head_object();
    // Метод для получения указателя на подчиненный объект по имени.
    cl_base* get_sub_object(string s_name);
    // Метод для вывода дерева объектов в консоль.
    void print_tree(int level);


    // Метод для подсчета количества объектов с заданным именем.
    int count(string s_name);
    // Метод для поиска объекта по имени.
    cl_base* search_object(string s_name);
    // Метод для поиска объекта с заданным именем относительно текущего
объекта.
    cl_base* find_object_from_current(string s_name);
    // Метод для поиска объекта с заданным именем относительно текущего или
подчиненных объектов.
    cl_base* find_object_from_current2(string s_name);
    // Метод для поиска объекта с заданным именем относительно корневого
объекта.
    cl_base* find_object_from_root(string s_name);
    // Метод для установки состояния объекта. Принимает номер состояния.
    void set_state(int new_state);
    // Метод, проверяющий, может ли объект быть готов (не имеет неготовых
подчиненных объектов).
    bool can_be_ready();
    // Метод для установки флага готовности объекта. Если state равен 0,
вызывает метод set_ready для всех подчиненных объектов.
    void set_ready(int state);
    // Метод для вывода дерева объектов и их готовности в консоль.
    void print_status(int level);
};

#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 17.

Таблица 17 – 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. 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).