

Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ.....	6
1.1 Описание входных данных.....	8
1.2 Описание выходных данных.....	9
2 МЕТОД РЕШЕНИЯ.....	12
3 ОПИСАНИЕ АЛГОРИТМОВ.....	15
3.1 Алгоритм метода find_obj_by_coord класса cl_base.....	15
3.2 Алгоритм метода delete_sub_obj класса cl_base.....	17
3.3 Алгоритм метода set_head_obj класса cl_base.....	18
3.4 Алгоритм функции main.....	19
3.5 Алгоритм конструктора класса cl_base.....	19
3.6 Алгоритм деструктора класса cl_base.....	20
3.7 Алгоритм метода get_name класса cl_base.....	21
3.8 Алгоритм метода get_p_head класса cl_base.....	21
3.9 Алгоритм метода set_name класса cl_base.....	21
3.10 Алгоритм метода get_subordinate_objects класса cl_base.....	22
3.11 Алгоритм метода search_object_from_current класса cl_base.....	23
3.12 Алгоритм метода search_object_from_tree класса cl_base.....	24
3.13 Алгоритм метода set_status класса cl_base.....	25
3.14 Алгоритм метода print_tree класса cl_base.....	26
3.15 Алгоритм метода print_status_tree класса cl_base.....	27
3.16 Алгоритм конструктора класса cl_application.....	28
3.17 Алгоритм метода build_tree_objects класса cl_application.....	28
3.18 Алгоритм метода exec_app класса cl_application.....	33
3.19 Алгоритм конструктора класса cl_2.....	34
3.20 Алгоритм конструктора класса cl_3.....	34
3.21 Алгоритм конструктора класса cl_4.....	35

3.22 Алгоритм конструктора класса cl_5.....	35
3.23 Алгоритм конструктора класса cl_6.....	35
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....	37
5 КОД ПРОГРАММЫ.....	58
5.1 Файл cl_2.cpp.....	58
5.2 Файл cl_2.h.....	58
5.3 Файл cl_3.cpp.....	58
5.4 Файл cl_3.h.....	59
5.5 Файл cl_4.cpp.....	59
5.6 Файл cl_4.h.....	59
5.7 Файл cl_5.cpp.....	60
5.8 Файл cl_5.h.....	60
5.9 Файл cl_6.cpp.....	60
5.10 Файл cl_6.h.....	61
5.11 Файл cl_application.cpp.....	61
5.12 Файл cl_application.h.....	64
5.13 Файл cl_base.cpp.....	64
5.14 Файл cl_base.h.....	70
5.15 Файл main.cpp.....	71
6 ТЕСТИРОВАНИЕ.....	72
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	75

1 ПОСТАНОВКА ЗАДАЧИ

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

Расширить функциональность базового класса:

- метод переопределения головного объекта для текущего в дереве иерархии. Метод должен иметь один параметр, указатель на объект базового класса, содержащий указатель на новый головной объект. Переопределение головного объект для корневого объекта недопустимо. Недопустимо создать второй корневой объект. Недопустимо при переопределении, чтобы у нового головного появились два подчиненных объекта с одинаковым наименованием. Новый головной объект не должен принадлежать к объектам из ветки текущего. Если переопределение выполнено, метод возвращает значение «истина», иначе «ложь»;
- метод удаления подчиненного объекта по наименованию. Если объект не найден, то метод завершает работу. Один параметр строкового типа, содержит наименование удаляемого подчиненного объекта;
- метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задаться в следующем виде:
 - o / - корневой объект;
 - o //«имя объекта» - поиск объекта по уникальной имени от корневого (для однозначности уникальность требуется в рамках дерева);
 - o . - текущий объект;
 - o .«имя объекта» - поиск объекта по уникальной имени от текущего (для однозначности уникальность требуется в рамках ветви дерева от

текущего объекта);

- о «имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;

- о /«имя объекта 1»[/«имя объекта 2»] . . . - абсолютная координата от корневого объекта.

Примеры координат:

```
/
//ob_3
.
.ob_2
ob_2/ob_3
/ob_1/ob_2/ob_3
```

Если координата - пустая строка или объект не найден или определяется неоднозначно (дуближ имен на ветке, на дереве), тогда вернуть нулевой указатель.

Наименование объекта не содержит символы «.» и «/».

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие. В строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему. При построении дерева уникальность наименования относительно множества непосредственно подчиненных объектов для любого головного объекта необходимо соблюдать. Если это требование исходя из входных данных нарушается, то соответствующий подчиненный объект не создается.

Добавить проверку допустимости исходной сборки. Собрать дерево невозможно, если по заданной координате головной объект не найден (например, ошибка в наименовании или еще не расположен на дереве объектов). Если номер класса объекта задан некорректно, то объект не создается.

Собранная система обрабатывает следующие команды:

- SET «координата» – устанавливает текущий объект;
- FIND «координата» – находит объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» задает новый головной объект;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим. При вводе данных в названии команд ошибок нет. Если при переопределении головного объекта нарушается уникальность наименований подчиненных объектов для нового головного, переопределение не производится.

1.1 Описание входных данных

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие. В строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

После ввода состава дерева иерархии построчно вводятся команды:

- SET «координата» – установить текущий объект;
- FIND «координата» – найти объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» соответствует новому главному объекту;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершить функционирование системы (выполнение программы).

Команды SET, FIND, MOVE и DELETE вводятся произвольное число раз.

Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов:

```
rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4 object_7 3
endtree
FIND object_2/object_4
SET /object_2
FIND //object_7
FIND object_4/object_7
FIND .
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
END
```

1.2 Описание выходных данных

Первая строка:

```
object tree
```

Со второй строки вывести иерархию построенного дерева как в работе версия №2.

При ошибке определения головного объекта, прекратить сборку, вывести иерархию уже построенного фрагмента дерева, со следующей строки сообщение:

```
The head object «координата головного объекта» is not found
```

и прекратить работу программы с кодом возврата 1.

Если при построении при попытке создания объекта обнаружен дубляж, то вывести:

«координата головного объекта» Dubbing the names of subordinate objects

Если дерево построено, то далее построчно вводятся команды.

Для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

The object was not found at the specified coordinate: «искомая координата объекта»

Для команд FIND вывести:

«искомая координата объекта» Object name: «наименование объекта»

Если объект не найден, то:

«искомая координата объекта» Object is not found

Для команд MOVE вывести:

New head object: «наименование нового головного объекта»

Если головной объект не найден, то:

«искомая координата объекта» Head object is not found

Если переопределить головной объект не удалось, то:

«искомая координата объекта» Redefining the head object failed

Если у нового головного объекта уже есть подчиненный с таким же именем,
то вывести:

«искомая координата объекта» Dubbing the names of subordinate objects

При попытке переподчинения головного объекта к объекту на ветке,
вывести:

«координата нового головного объекта» Redefining the head object failed

Для команды DELETE:

Если подчиненный объект удален, то вывести:

The object «абсолютный путь удаленного объекта» has been deleted

Если объект не найден, то ничего не выводить.

После команды END с новой строки вывести:

Current object hierarchy tree

Со следующей строки вывести текущую иерархию дерева.

Пример вывода иерархии дерева объектов:

```
Object tree
rootela
  object_1
    object_7
  object_2
    object_4
      object_7
    object_5
    object_3
  object_3
object_2/object_4    Object name: object_4
Object is set: object_2
//object_7    Object is not found
object_4/object_7    Object name: object_7
.    Object name: object_2
.object_7    Object name: object_7
object_4/object_7    Object name: object_7
.object_7    Redefining the head object failed
Object is set: object_7
//object_1    Dubbing the names of subordinate objects
New head object: object_3
Current object hierarchy tree
rootela
  object_1
    object_7
  object_2
    object_4
      object_5
      object_3
  object_3
    object_7
```

2 МЕТОД РЕШЕНИЯ

Класс `cl_base`:

- свойства/поля:
 - поле установление наименования у объекта:
 - наименование — `s_object_name`;
 - тип — `string`;
 - модификатор доступа — `private`;
 - поле указатель на головной объект:
 - наименование — `p_head_object`;
 - тип — `cl_base*`;
 - модификатор доступа — `private`;
 - поле динамический массив указателей на подчиненные объекты:
 - наименование — `subordinate_objects`;
 - тип — `vector<cl_base*>`;
 - модификатор доступа — `private`;
 - поле индикатор состояния объекта:
 - наименование — `status`;
 - тип — `int`;
 - модификатор доступа — `private`;
- функционал:
 - метод `find_obj_by_coord` — метод поиска объекта по заданной координате;
 - метод `delete_sub_obj` — метод удаления подчиненного объекта по имени;
 - метод `set_head_obj` — метод смены головного элемента;
 - метод `cl_base` — параметризованный конструктор;

- o метод `~cl_base` — деструктор;
- o метод `get_name` — метод получения имени объекта;
- o метод `get_p_head` — метод получения указателя на родительский объект;
- o метод `set_name` — метод установки нового имени объекта;
- o метод `get_subordinate_objects` — получение указателя на подчиненные объект по имени;
- o метод `search_object_from_current` — поиск объекта на ветке иерархии от текущего по имени;
- o метод `search_object_from_tree` — поиск объекта по дереву;
- o метод `set_status` — метод установки статуса объекта;
- o метод `print_tree` — метод вывода дерева;
- o метод `print_status_tree` — метод вывода состояния объектов;
- o метод `set_head_obj` — метод смены головного объекта.

Класс `cl_application`:

- функционал:
 - o метод `cl_application` — параметризованный конструктор;
 - o метод `build_tree_objects` — метод создает иерархию объектов;
 - o метод `exec_app` — метод запуска приложения.

Класс `cl_2`:

- функционал:
 - o метод `cl_2` — параметризованный конструктор.

Класс `cl_3`:

- функционал:
 - o метод `cl_3` — параметризованный конструктор.

Класс `cl_4`:

- функционал:

- о метод cl_4 — параметризированный конструктор.

Класс cl_5:

- функционал:
 - о метод cl_5 — параметризированный конструктор.

Класс cl_6:

- функционал:
 - о метод cl_6 — параметризированный конструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			базовый класс	
		cl_applicatio n	public		2
		cl_2	public		3
		cl_3	public		4
		cl_4	public		5
		cl_5	public		6
		cl_6	public		7
2	cl_applicatio n			класс приложения	
3	cl_2				
4	cl_3				
5	cl_4				
6	cl_5				
7	cl_6				

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 `find_obj_by_coord` класса `cl_base`

Функционал: метод поиска объекта по заданной координате.

Параметры: `string s_coord` - координата искомого объекта.

Возвращаемое значение: `cl_base*`.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода `find_obj_by_coord` класса `cl_base`

№	Предикат	Действия	№ перехода
1		<code>p_root</code> присваивается указатель на текущий объект Инициализация <code>i_slash_2 = 0</code> , для хранения индекса 2-ого слэша Строковый тип <code>s_name</code> , для хранения пути объекта	2
2	<code>s_coord</code> равен <code>"/"</code>		3
			4
3	<code>p_root</code> не равен пустому указателю	<code>p_root</code> присваивается значение вызываемого метода <code>get_p_head()</code> от <code>p_root</code>	3
		возвращаем <code>p_root</code>	Ø
4	<code>s_coord</code> равен <code> "."</code>	возвращаем текущий объект	Ø
			5
5	<code>s_coord[0]</code> равен <code>"/"</code> и <code>s_coord[1]</code> равен <code>"/"</code>	от текущего объекта вызывается метод <code>search_object_from_tree</code> с параметром <code>s_coord</code> с встроенным методом <code>substr(2)</code> (начиная с 3 символа)	Ø

№	Предикат	Действия	№ перехода
			6
6	s_coord[0] равен "."	от текущего объекта вызывается метод search_object_from_tree с параметром s_coord с встроенным методом substr(1) (начиная со 2 символа)	∅
			7
7		i_slah_2 присваивается индекс второго слэша, если он не найден он равен -1	8
8	s_coord[0] равен "/"		9
			14
9	головной объект не равен пустому указателю	p_root присваивается значение вызываемого метода get_p_head() от p_root	9
			10
10	i_slah_2 не равен -1	s_name присваивается значение объекта до следующего слэша из переменной s_coord	11
		s_name присваивается значение объекта с 1 символа s_coord	13
11		p_obj присваивается значение вызываемого метода get_subordinate_object с параметром s_name объекта p_root	12
12	p_obj не равен пустому указателю	возвращаем вызываем метод find_obj_by_coord с параметром s_coord (отсекается строка от найденного индекса слэша) объекта p_obj	∅
		возвращаем p_obj	∅
13		возвращаем вызываемый метод get_subordinate_object с параметром s_name объекта p_root	∅
14	i_slah_2 не равен -1	s_name присваивается значение объекта до следующего слэша из переменной s_coord	15

№	Предикат	Действия	№ перехода
		возвращаем вызываемый метод get_subordinate_object с параметром s_coord текущего объекта	∅
15		p_obj присваивается значение вызываемого метода get_subordinate_object с параметром s_name текущего объекта	16
16	p_obj не равен пустому указателю	возвращаем вызываемый метод find_obj_by_coord с параметром s_coord (строка, после найденного первого слэша) объекта p_obj	∅
		возвращаем p_obj	∅

3.2 Алгоритм метода delete_sub_obj класса cl_base

Функционал: метод удаления подчиненного объекта по имени.

Параметры: string s_name - имя удаляемого объекта.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода delete_sub_obj класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация i = длины subordinates_objects -1	2
2	i >= 0		3
			∅
3	i-ым элемент subordinates_objects с полученным именем s_object_name равен s_name	удаляем этот i-ый элемент subordinate_objects	∅
			4
4		--i	2

3.3 Алгоритм метода set_head_obj класса cl_base

Функционал: метод смены головного элемента.

Параметры: cl_base * p_new_head - имя нового головного объекта .

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_head_obj класса cl_base

№	Предикат	Действия	№ перехода
1		создаем дуюликат указатель p_temp = p_new_head	2
2	p_new_head равен нулевому указателю	возвращаем false	∅
			3
3	корневой элемент равен нулевому указателю	возвращаем false	∅
			4
4	у p_new_head есть подчиненные с одинаковым именем	возвращаем false	∅
			5
5	p_temp не равен пустому множеству		6
			8
6	p_temp равен текущему объекту	возвращаем false	∅
			7
7		p_temp присваивается значение p_head_object p_temp	5
8		у текущего головного объекта удаляются подчиненные объекты	9

№	Предикат	Действия	№ перехода
9		присваиваем p_head_object текущего объекта p_new_head	10
10		p_new_head добавляется подчиненные объекты текущего объекта	11
11		возвращаем true	∅

3.4 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 5.

Таблица 5 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_cl_application класса cl_application в качестве параметра подается нулевой указатель	2
2		вызов метода build_tree_objects объекта ob_cl_application	3
3		вызов метода exes_app объекта ob_cl_application	∅

3.5 Алгоритм конструктора класса cl_base

Функционал: параметризованный конструктор.

Параметры: cl_base* p_head_object - указатель на головной объект,
strings_object_name - имя узла дерева.

Алгоритм конструктора представлен в таблице 6.

Таблица 6 – Алгоритм конструктора класса *cl_base*

№	Предикат	Действия	№ перехода
1		полю <i>p_head_object</i> этого объекта присваивается параметр <i>p_head_object</i>	2
2		полю <i>s_object_name</i> этого объекта присваивается параметр <i>s_object_name</i>	3
3	<i>p_head_object</i> ненулевой	в вектор указателей подчиненных объектов головного объекта добавляется указатель на этот объект	∅
			∅

3.6 Алгоритм деструктора класса *cl_base*

Функционал: деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 7.

Таблица 7 – Алгоритм деструктора класса *cl_base*

№	Предикат	Действия	№ перехода
1		указатель <i>p_sub</i> настраиваем на 1-ый объект <i>subordinate_object</i>	2
2	<i>p_sub</i> не указывает на конец <i>subordinate_objects</i>	удаляем объект по адресу <i>p_sub</i>	3
			∅
3		указатель <i>p_sub</i> настраиваем на следующий элемент в <i>subordinate_objects</i>	2

3.7 Алгоритм метода `get_name` класса `cl_base`

Функционал: метод получения имени объекта.

Параметры: нет.

Возвращаемое значение: `string`.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода `get_name` класса `cl_base`

№	Предикат	Действия	№ перехода
1		возврат <code>s_object_name</code>	Ø

3.8 Алгоритм метода `get_p_head` класса `cl_base`

Функционал: метод получения указателя на родительский объект.

Параметры: нет.

Возвращаемое значение: `cl_base*`.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода `get_p_head` класса `cl_base`

№	Предикат	Действия	№ перехода
1		возврат <code>p_head_object</code>	Ø

3.9 Алгоритм метода `set_name` класса `cl_base`

Функционал: метод установки нового имени объекта.

Параметры: `string new_name`.

Возвращаемое значение: `bool`.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода *set_name* класса *cl_base*

№	Предикат	Действия	№ перехода
1	p_head_object этого объекта не равен пустому указателю		2
		s_object_name текущего объекта присваивается new_name	5
2		указатель p_sub настраиваем на 1-ый subordinate_objects элемент	3
3	переменная p_sub вектора subordinate_objects указателя p_head_objects не равна nullptr		4
			5
4	результат вызова метода get_name объекта p_sub равен new_name	возврат false	∅
		p_sub присваивается следующий элемент subordinate_objects	3
5		возврат true	∅

3.10 Алгоритм метода *get_subordinate_objects* класса *cl_base*

Функционал: получение указателя на подчиненные объект по имени.

Параметры: string search_name.

Возвращаемое значение: *cl_base**.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода *get_subordinate_objects* класса *cl_base*

№	Предикат	Действия	№ перехода
1		инициализация $i = 0$	2
2	$i < \text{длины subordinate_objects}$		3
			4
3	результат вызова <i>get_name</i> i -ого элемента <i>subordinate_objects</i> равен <i>search_name</i>	возврат i -ого элемента <i>subordinate_objects</i>	∅
		$i++$	2
4		возврат <i>nullptr</i>	∅

3.11 Алгоритм метода *search_object_from_current* класса *cl_base*

Функционал: поиск объекта на ветке иерархии от текущего по имени.

Параметры: *string s_name*.

Возвращаемое значение: *cl_base**.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода *search_object_from_current* класса *cl_base*

№	Предикат	Действия	№ перехода
1		объявление очереди <i>q</i> , которая принимает указатели на объекты класса <i>cl_base</i>	2
2		инициализация пустого указателя <i>p_found</i> на объекты класса <i>cl_base</i>	3
3		добавление в конец очереди <i>q</i> указателя на	4

№	Предикат	Действия	№ перехода
		текущий объект	
4	очередь q не пустая	инициализация указателя p_front на объект класса cl_base значением первого элемента в очереди q	5
			11
5		удаление первого элемента очереди q	6
6	имя объекта по указателю p_front равно параметру s_name		7
			8
7	p_found ненулевой	возврат нулевого указателя	∅
		присваивание p_found значение p_front	8
8		указатель p_sub настраиваем на 1-ый элемент в subordinate_objects объекта с указателем p_front	9
9	p_sub не указывает на конец subordinate_objects	добавление в конец очереди q элемента p_sub	10
			4
10		указатель p_sub настраиваем на следующий элемент subordinate_objects	9
11		возврат p_found	∅

3.12 Алгоритм метода search_object_from_tree класса cl_base

Функционал: поиск объекта по дереву.

Параметры: string s_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода search_object_from_tree класса cl_base

№	Предикат	Действия	№ перехода
1		p_root присваивается указатель на текущий объект	2
2	вышестоящий объект p_root в дереве не пустой	p_root присваивается имя головного объекта	1
		возврат результат вызова метода search_object_from_current с параметром s_name	∅

3.13 Алгоритм метода set_status класса cl_base

Функционал: метод установки статуса объекта.

Параметры: int status.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода set_status класса cl_base

№	Предикат	Действия	№ перехода
1	p_head_objects нулевой или поле status родительского объекта != 0	присвоение скрытому полю текущего объекта status параметра status	2
			2
2	status равен 0	присвоение скрытому полю текущего объекта status параметра status	3

№	Предикат	Действия	№ перехода
			∅
3		инициализация $i = 0$	4
4	$i < \text{размера вектора subordinate_objects}$	вызов метода <code>set_status</code> с параметром <code>status</code> i -ого объекта <code>subordinate_objects</code>	5
			∅
5		$i++$	4

3.14 Алгоритм метода `print_tree` класса `cl_base`

Функционал: метод вывода дерева.

Параметры: `int layer`.

Возвращаемое значение: `void`.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода `print_tree` класса `cl_base`

№	Предикат	Действия	№ перехода
1		вывод со следующей строки, вывод <code>layer</code> пробелов $<<$ результат вызова метода <code>get_name</code> текущего объекта	2
2		настраиваем <code>p_sub</code> на 1-ый элемент <code>subordinate_objects</code>	3
3	<code>p_sub</code> не указывает на конец <code>subordinate_objects</code>	вызов метода <code>print_tree</code> объекта <code>p_sub</code> с параметром <code>layer+4</code>	4
			∅
4		указатель <code>p_sub</code> присваивается следующий	3

№	Предикат	Действия	№ перехода
		элемент subordinate_objects	

3.15 Алгоритм метода print_status_tree класса cl_base

Функционал: метод вывода состояния объектов.

Параметры: int layer.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода print_status_tree класса cl_base

№	Предикат	Действия	№ перехода
1		вывод переноса строки	2
2		инициализация i = 0	3
3	i < layer	вывод 4-ех пробелов	4
			5
4		i++	3
5	status ненулевой	вывод имени текущего объекта и "is ready"	6
		вывод имени текущего объекта и "is not ready"	6
6		инициализация i = 0	7
7	i < размера вектора subordinate_objects	i < размера вектора subordinate_objвызов метода print_status_tree с параметром layer+1 i-ого объекта subordinate_objectsects	8
			∅
8		i++	7

3.16 Алгоритм конструктора класса `cl_application`

Функционал: параметризированный конструктор.

Параметры: нет.

Алгоритм конструктора представлен в таблице 17.

Таблица 17 – Алгоритм конструктора класса `cl_application`

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса <code>cl_base</code> с параметром <code>p_head_objects</code>	Ø

3.17 Алгоритм метода `build_tree_objects` класса `cl_application`

Функционал: метод создает иерархию объектов.

Параметры: нет.

Возвращаемое значение: `void`.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода `build_tree_objects` класса `cl_application`

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2
2		объявление строковых переменных <code>s_head_name</code> и <code>s_sub_name</code> , объявление <code>int class_number</code> , <code>object_state</code>	3
3		инициализация указателя <code>p_head</code> на объект класса <code>cl_base</code> указателем на текущий объект	4

№	Предикат	Действия	№ перехода
4		ввод значения s_head_name	5
5		вызов метода set_name с параметром s_head_name текущего объекта	6
6		ввод значения s_head_name	7
7	s_head_name не равен "endtree"		8
			18
8		ввод значение s_sub_name и class_number	9
9		присвоение объекту p_head результат работы метода search_object_from_tree с параметром s_head_name	10
10	p_head равен нулевому указателю	вызов метода print_tree	11
			12
11		вывод с новой строки "The head object " << s_head_name << " is not found"	∅
12	class_number равен 2	создание объекта класса cl_2 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			13
13	class_number равен 3	создание объекта класса cl_3 с параметрами	6

№	Предикат	Действия	№ перехода
		p_head_objects и s_sub_name с помощью оператора new	
			14
14	class_number равен 4	создание объекта класса cl_4 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			15
15	class_number равен 5	создание объекта класса cl_5 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			16
16	class_number равен 6	создание объекта класса cl_6 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			17
17		ввод значения s_head_name	18
18		вызов от текущего объекта метод print_tree	19
19		p_current присваивается указатель на текущий объект объявление строковых переменных s_command, s_coordinate Объявление указателя p_ob	20
20		ввод значения для s_command	21
21	s_command равен "END"		∅
			22

№	Предикат	Действия	№ перехода
22		ввод значения для s_coordinate	23
23		p_obj присваивается значение метода find_obj_by_coord с параметром s_coordinate объекта p_current	24
24	s_command равен "SET"		25
			27
25	p_obj не равен пустому указателю	p_current присваивается значение p_obj	26
		вывод переноса строки << "The object was not found at the specified coordinate: " << s_coordinate	27
26		вывод переноса строки << "Object is set: " << вызов метода get_name объекта p_current	27
27	s_command равен "FIND"		28
			29
28	p_obj не равен пустому указателю	вывод переноса строки << s_coordinate << " Object name: " << вызов метода get_name объекта p_obj	29
		вывод переноса строки << s_coordinate << " Object is not found"	29
29	s_command равен "MOVE"		30
			35
30	метод set_head_obj с параметром p_obj объекта	вывод << перенос строки << "New head object: " << вызов метода get_name объекта p_obj	35

№	Предикат	Действия	№ перехода
	p_current вернет true		
			31
31		указатель p_temp (дубликат) присваивается значение p_obj	32
32	p_obj равен пустому указателю	вывод переноса строки << s_coordinate << "Head object is not found"	33
			33
33	у головного объекта p_obj есть подчиненные объекты с похожим именем p_obj	вывод переноса строки <<" Dubbing the names of subordinate objects"	34
			34
34	вызываемый метод get_p_head текущего объекта равен пустому указателю	вывод переноса строки << s_coordinate << "Redefining the head object failed"	35
			35
35	s_command равен "DELETE"		36
			42
36	p_obj не равен пустому указателю	объявляется строковая переменная s_abs_path, для хранения абсолютного пути объекта объявляется указатель дубликат p_object хранящий значение p_obj	37
			42
37	головной объект p_object не	s_abs_path присваивается "/" + get_name() объекта	38

№	Предикат	Действия	№ перехода
	равен пустому указателю	p_object + s_abs_path	
			39
38		p_object присваивается значение метода get_p_head объекта p_object	39
39		вывод переноса строки << "The object " << s_abs_path << " has been deleted"	40
40		от текущего объекта p_current вызывается метод delete_sub_obj с параметром p_obj->get_name() для удаления подчиненных объектов	41
41		деструктор найденного объекта p_obj	42
42		ввод значения s_command	21

3.18 Алгоритм метода exes_app класса cl_application

Функционал: метод запуска приложения.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода exes_app класса cl_application

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Current object hierarchy tree"	2
2		вызов метода print_tree текущего объекта	3

№	Предикат	Действия	№ перехода
3		возврат 0	∅

3.19 Алгоритм конструктора класса cl_2

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 20.

Таблица 20 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head_objects и s_object_name	∅

3.20 Алгоритм конструктора класса cl_3

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 21.

Таблица 21 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head_objects и s_object_name	∅

3.21 Алгоритм конструктора класса cl_4

Функционал: параметризованный конструктор.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 22.

Таблица 22 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		вызов параметризованного конструктора класса cl_base с параметрами p_head_objects и s_object_name	Ø

3.22 Алгоритм конструктора класса cl_5

Функционал: параметризованный конструктор.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 23.

Таблица 23 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		вызов параметризованного конструктора класса cl_base с параметрами p_head_objects и s_object_name	Ø

3.23 Алгоритм конструктора класса cl_6

Функционал: параметризованный конструктор.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 24.

Таблица 24 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		вызов параметризованного конструктора класса cl_base с параметрами p_head_objects и s_object_name	Ø

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-21.

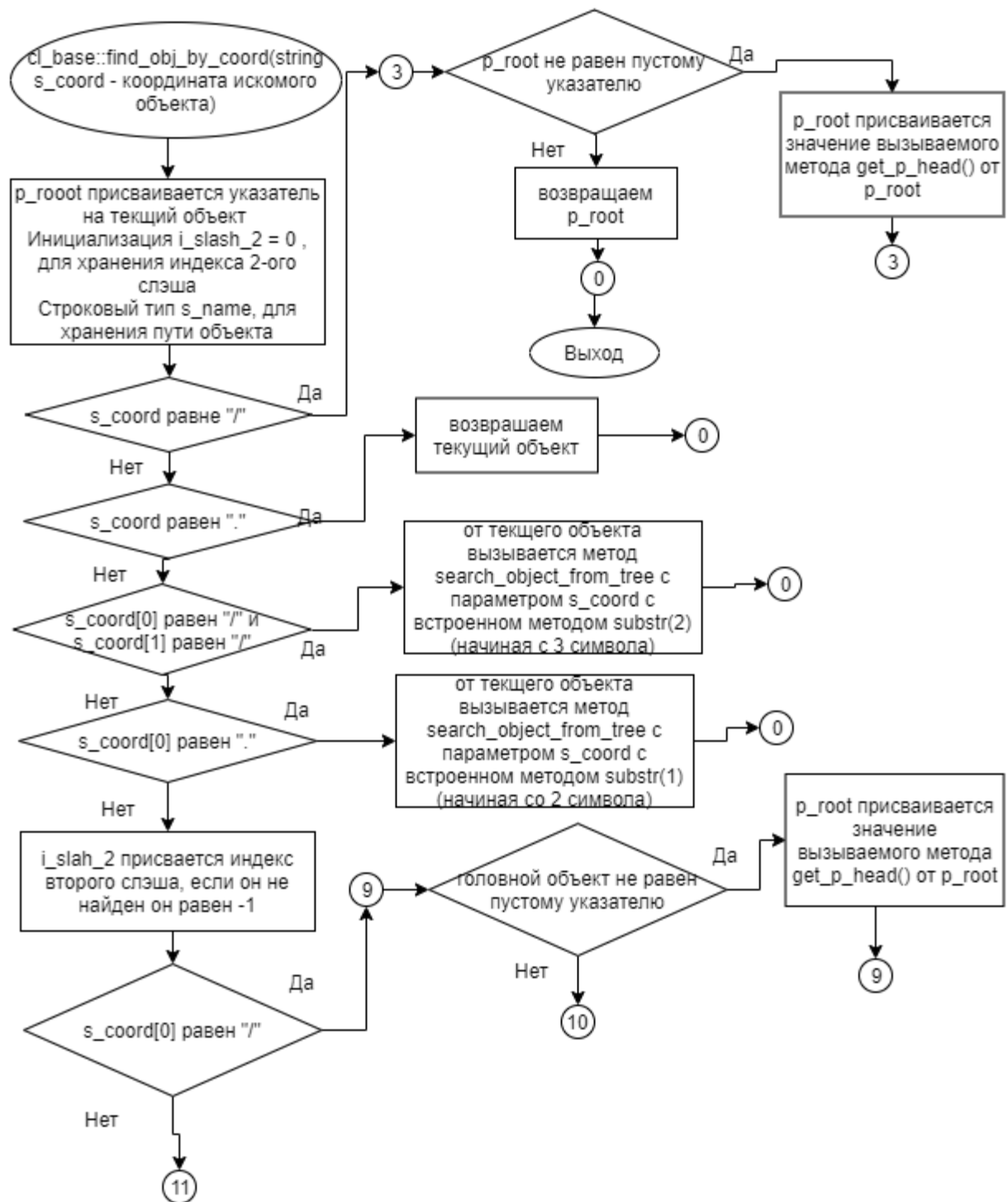


Рисунок 1 – Блок-схема алгоритма

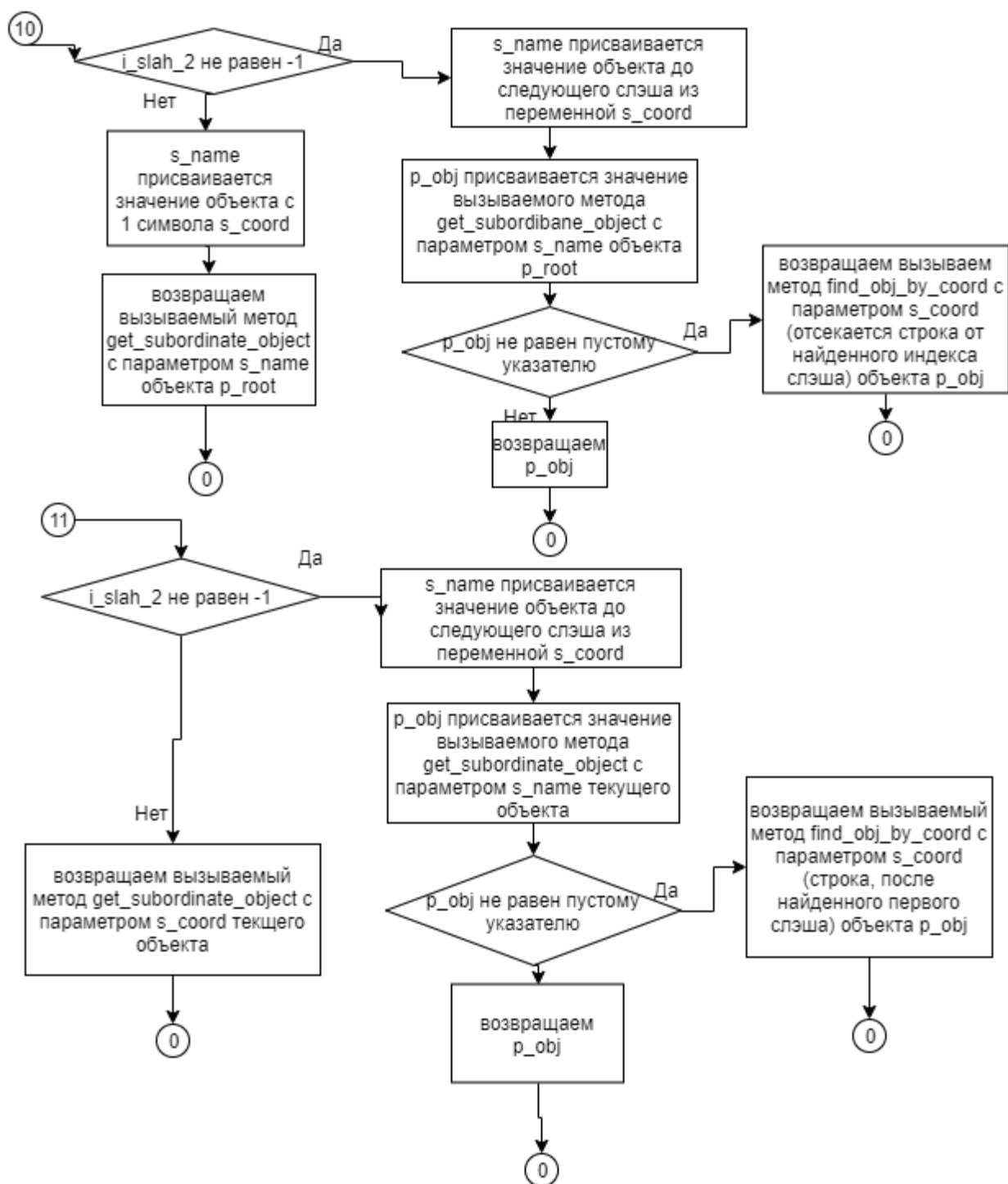


Рисунок 2 – Блок-схема алгоритма

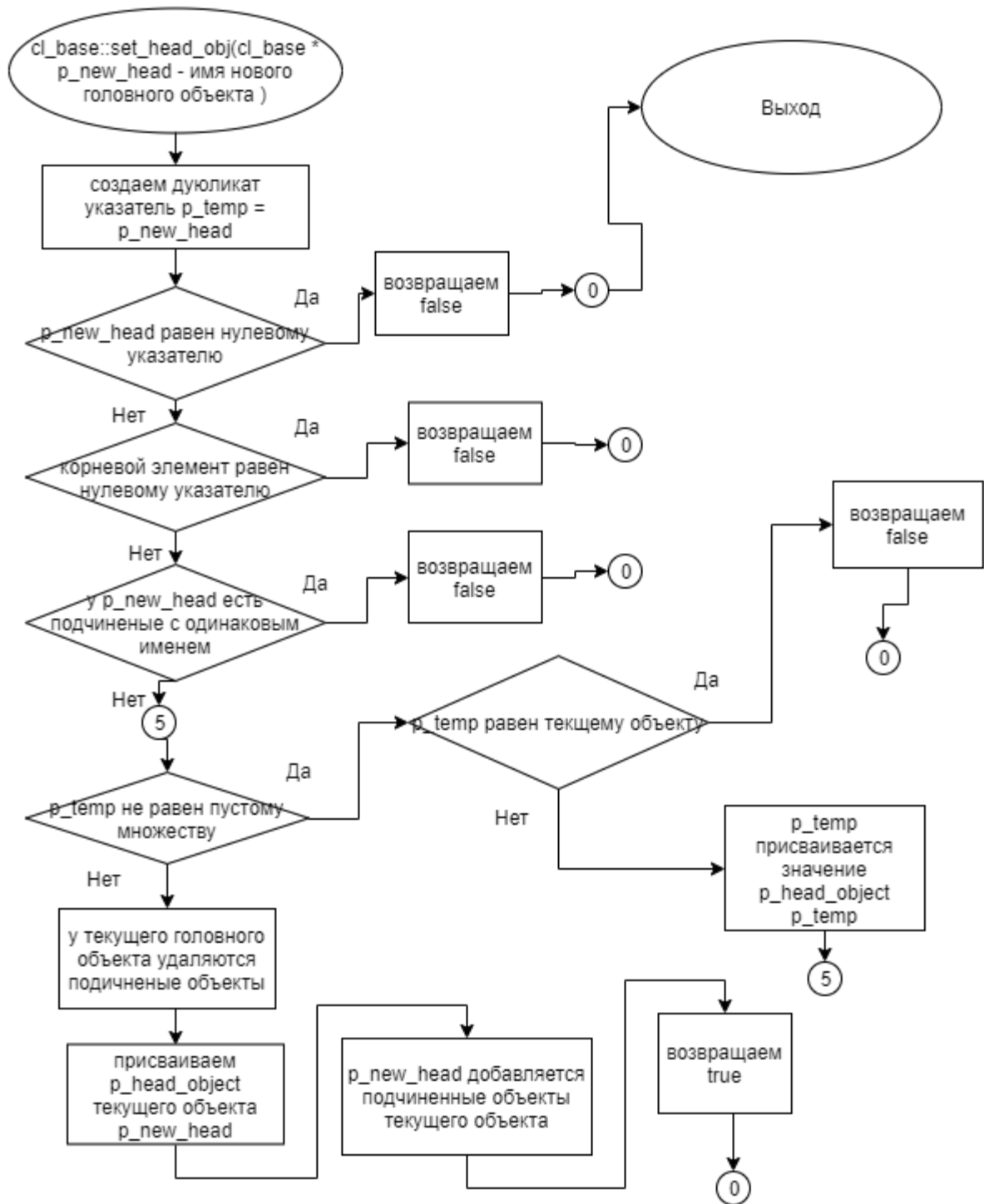


Рисунок 3 – Блок-схема алгоритма

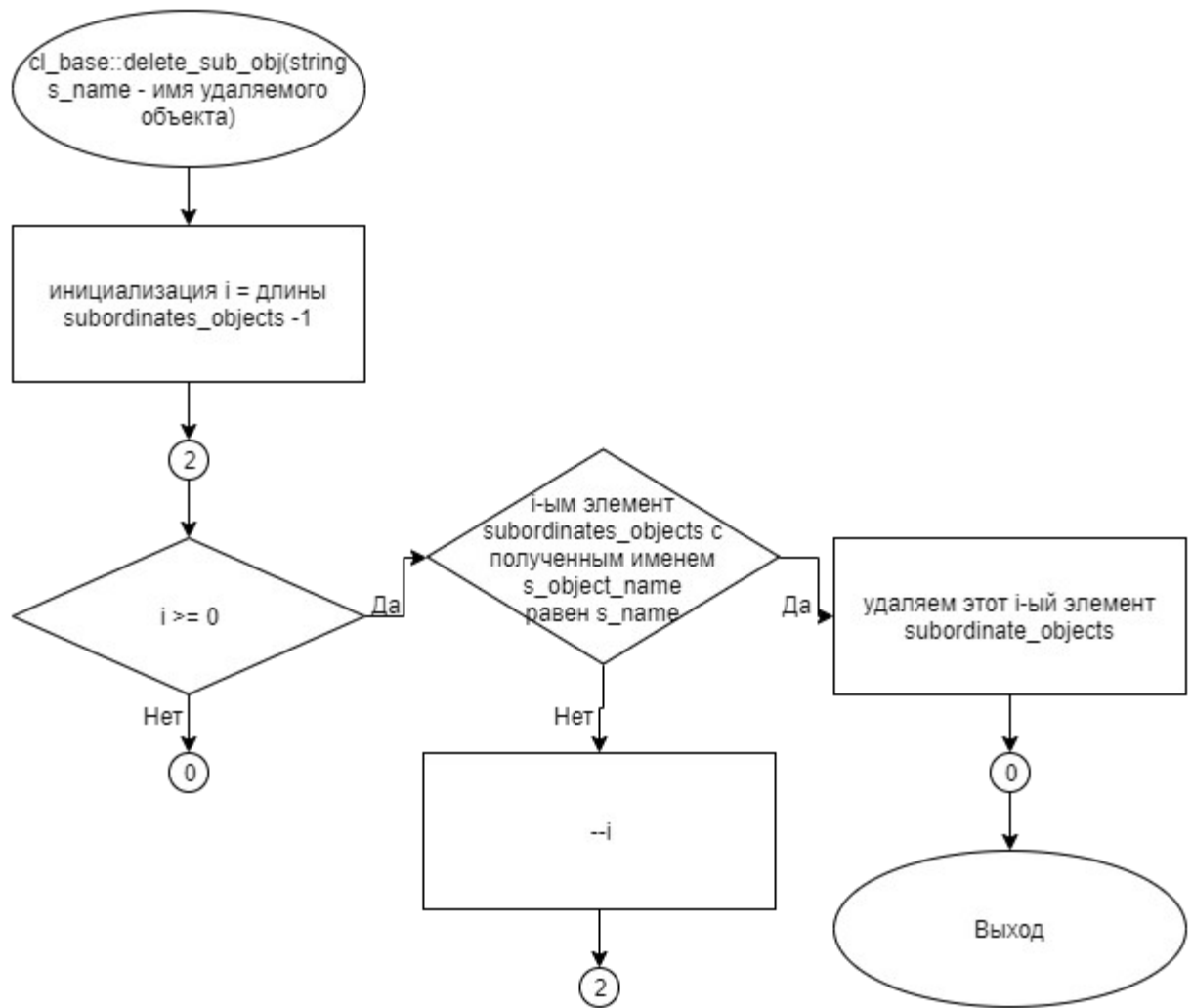


Рисунок 4 – Блок-схема алгоритма

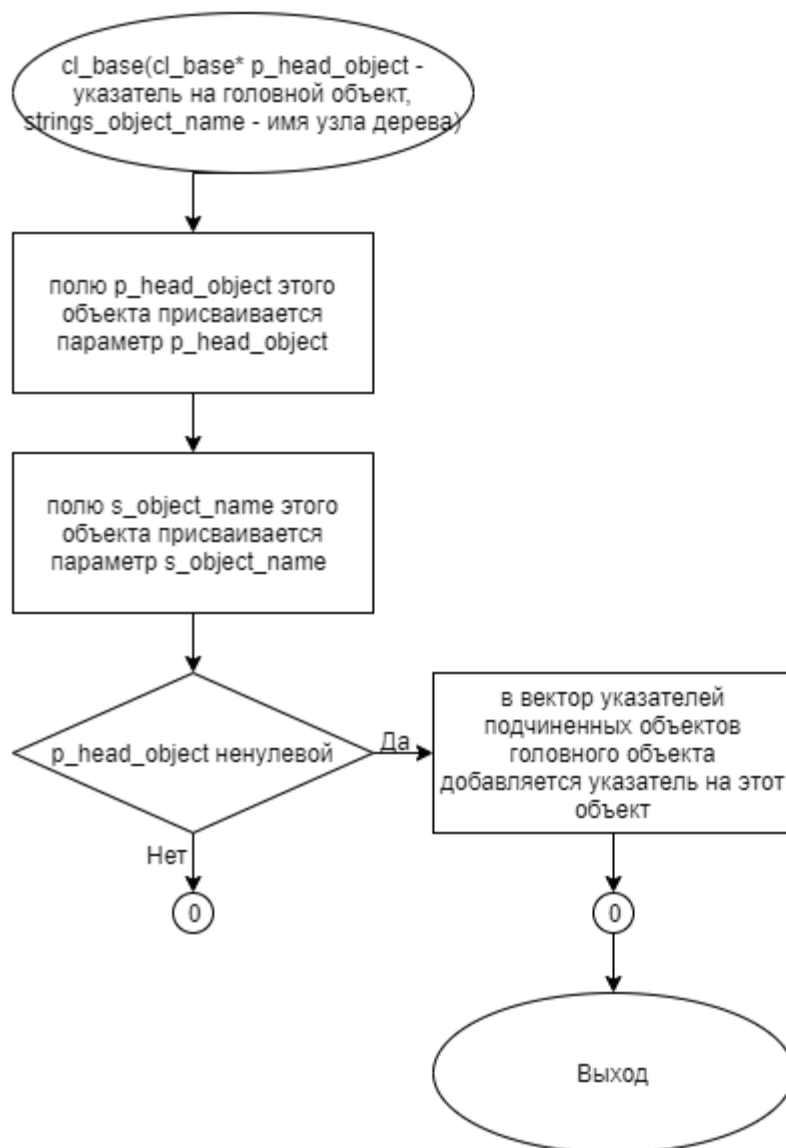


Рисунок 5 – Блок-схема алгоритма

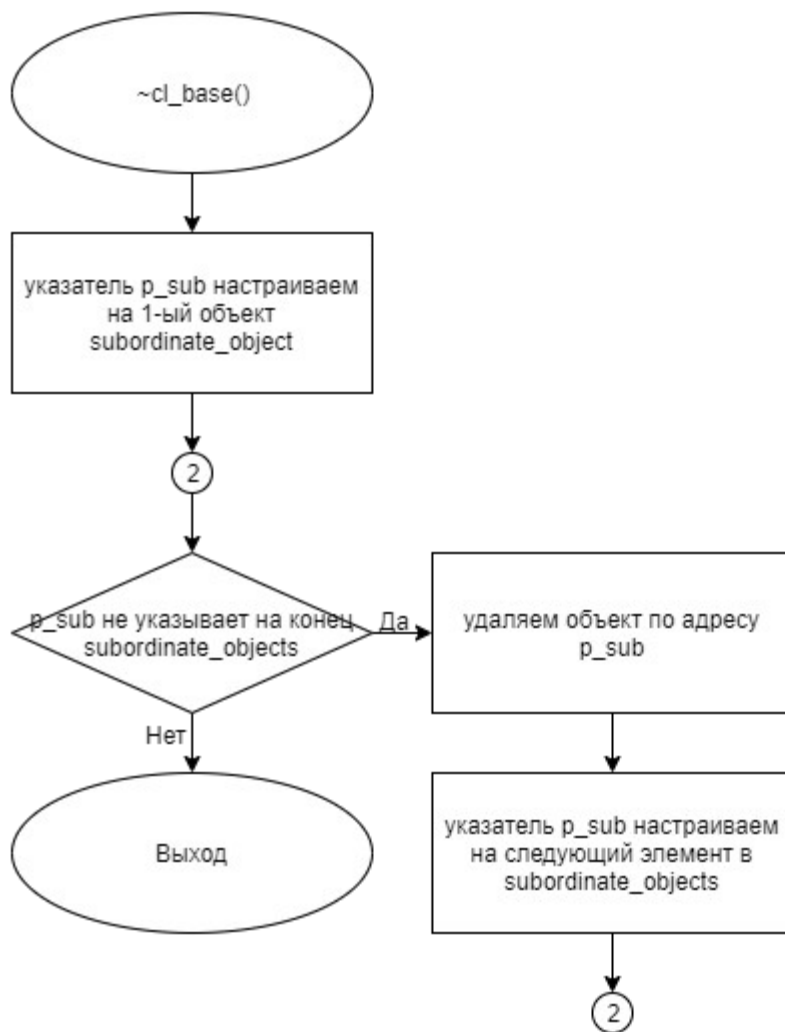


Рисунок 6 – Блок-схема алгоритма

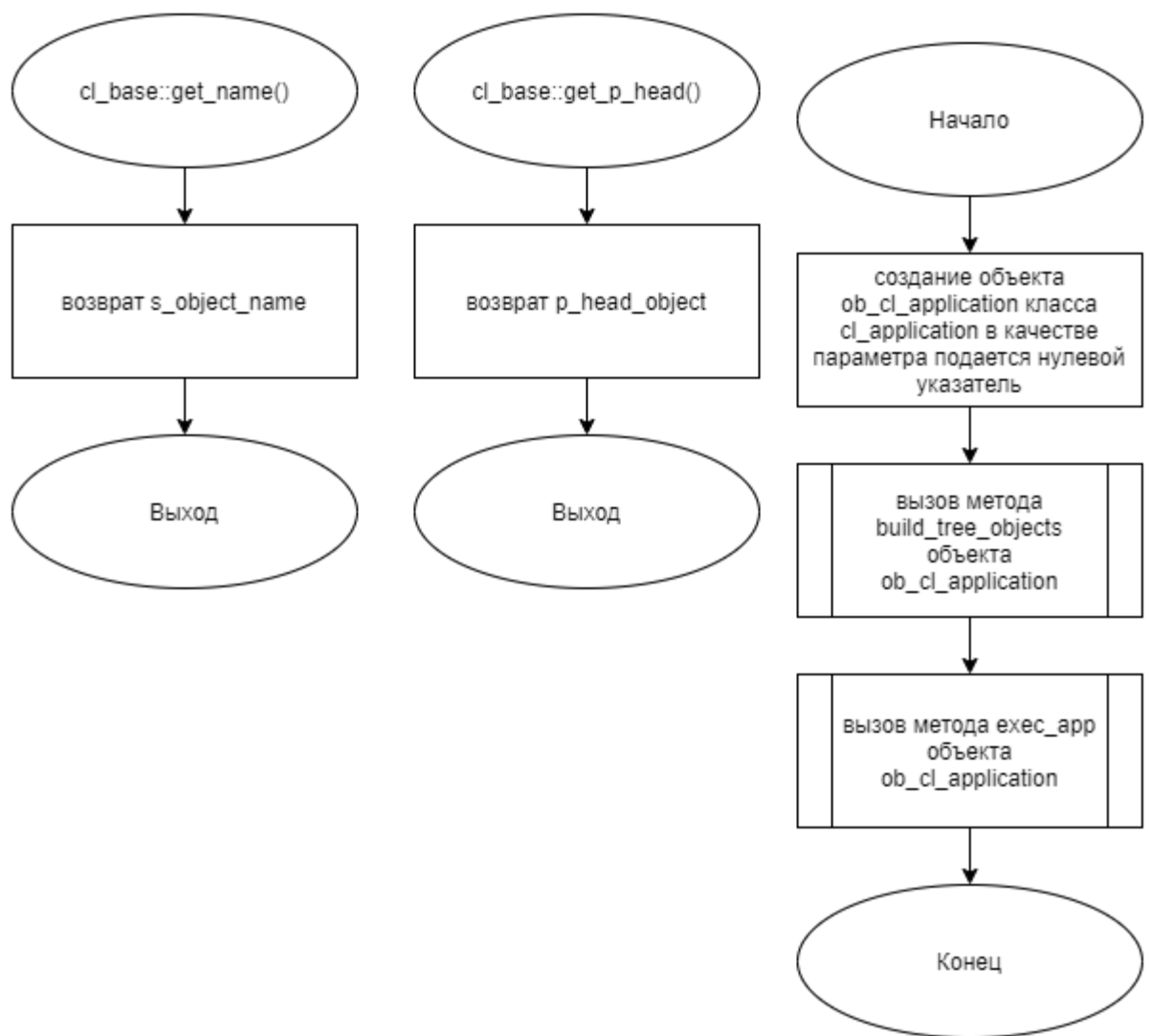


Рисунок 7 – Блок-схема алгоритма

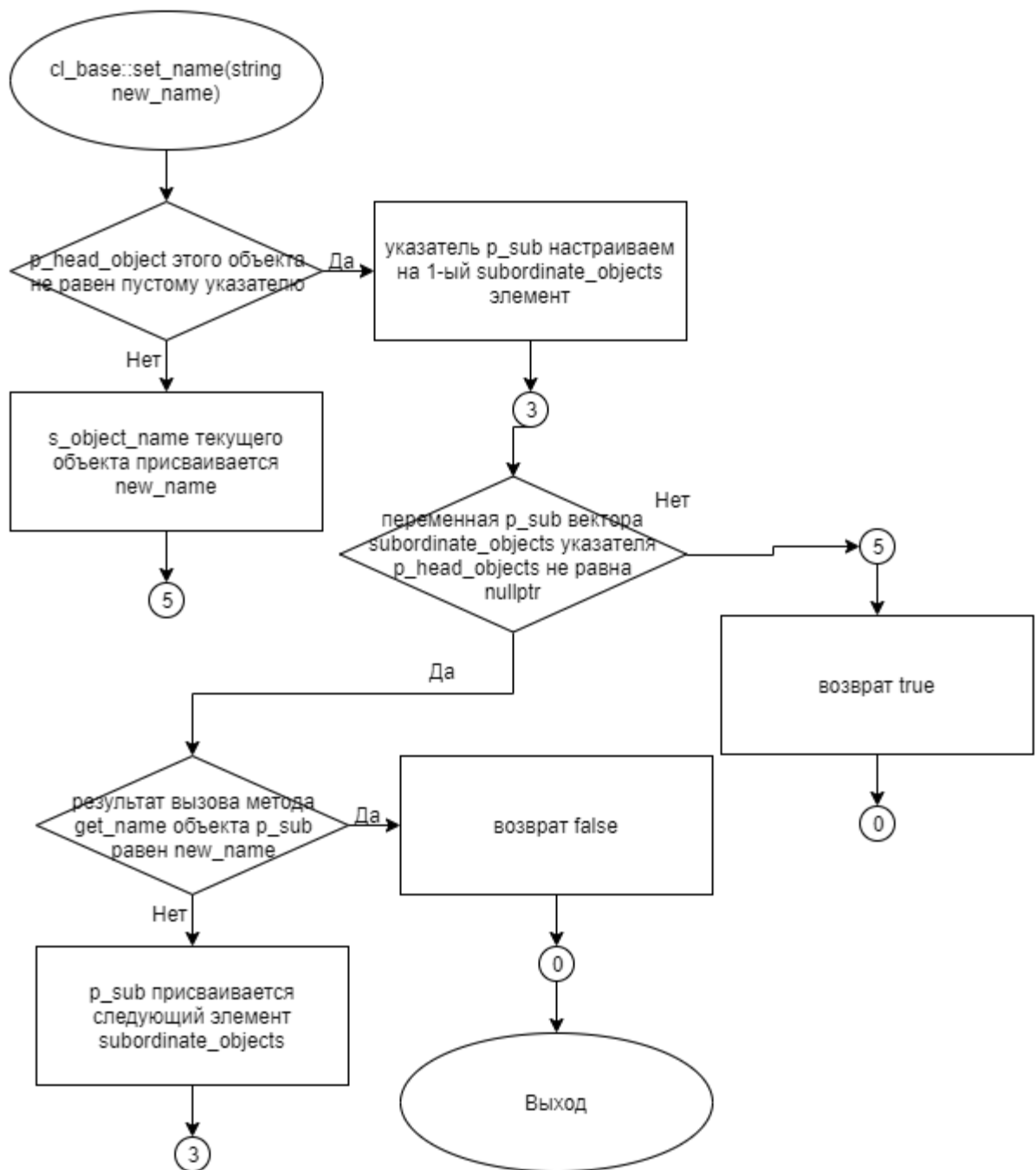


Рисунок 8 – Блок-схема алгоритма

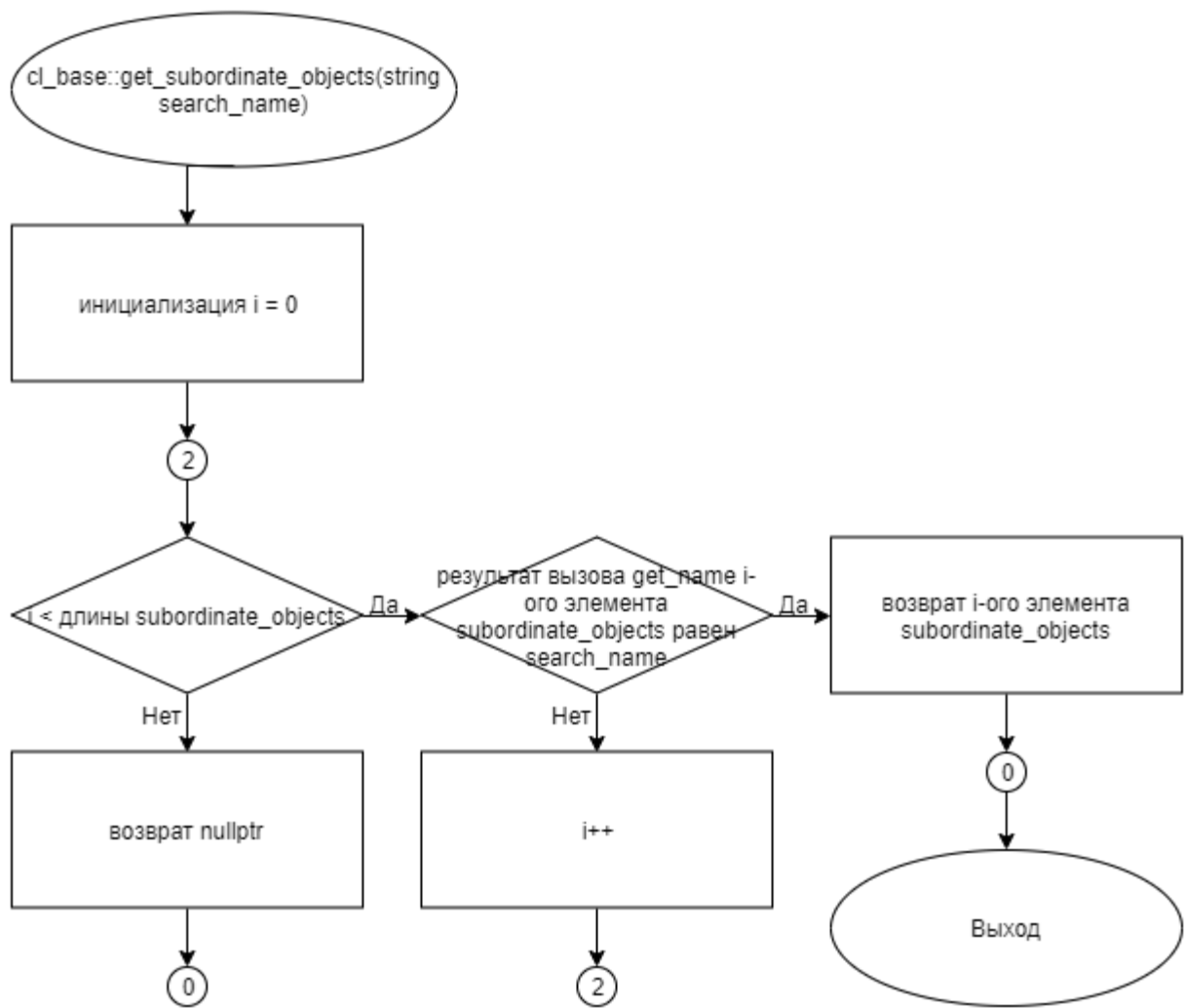


Рисунок 9 – Блок-схема алгоритма

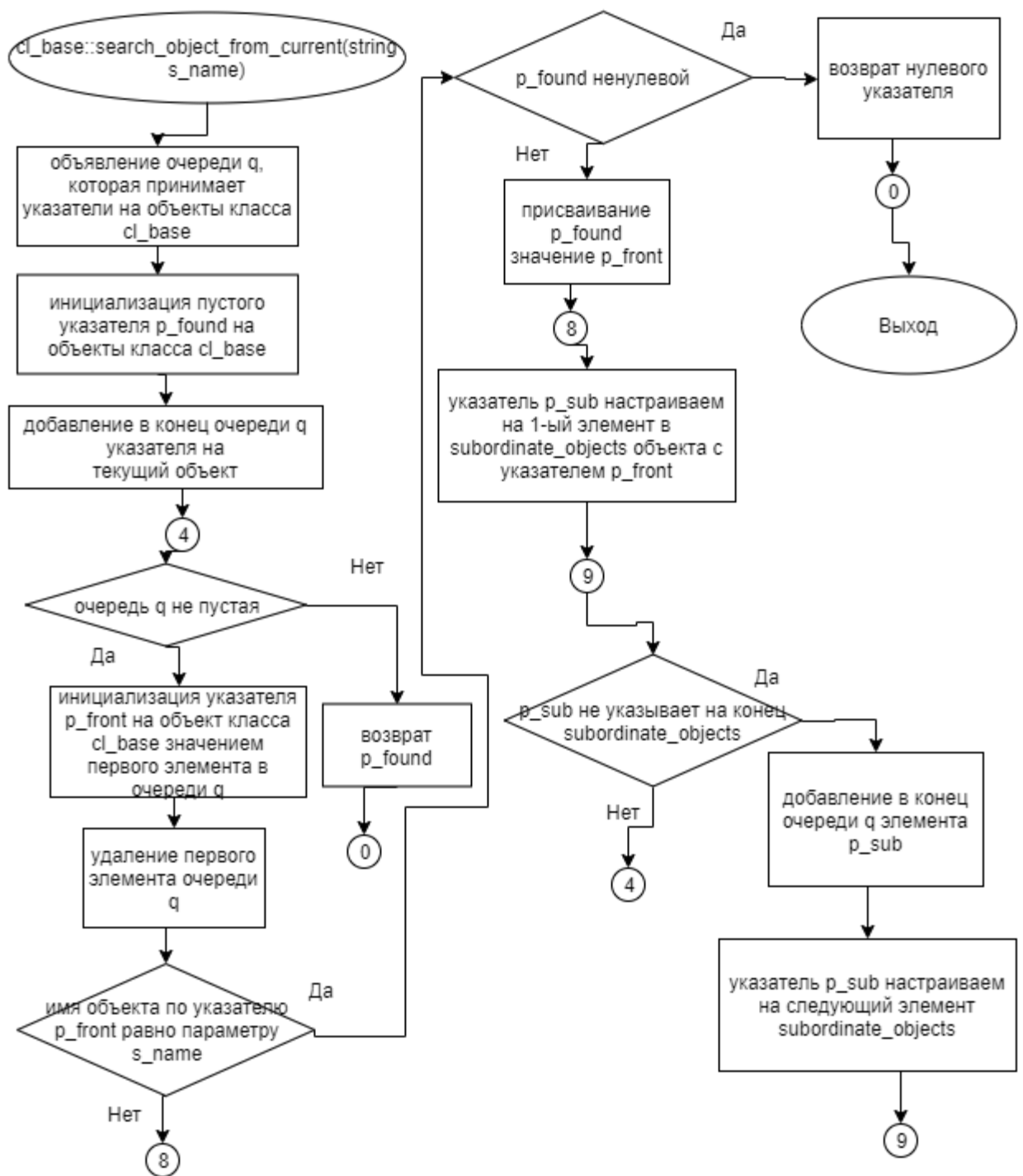


Рисунок 10 – Блок-схема алгоритма

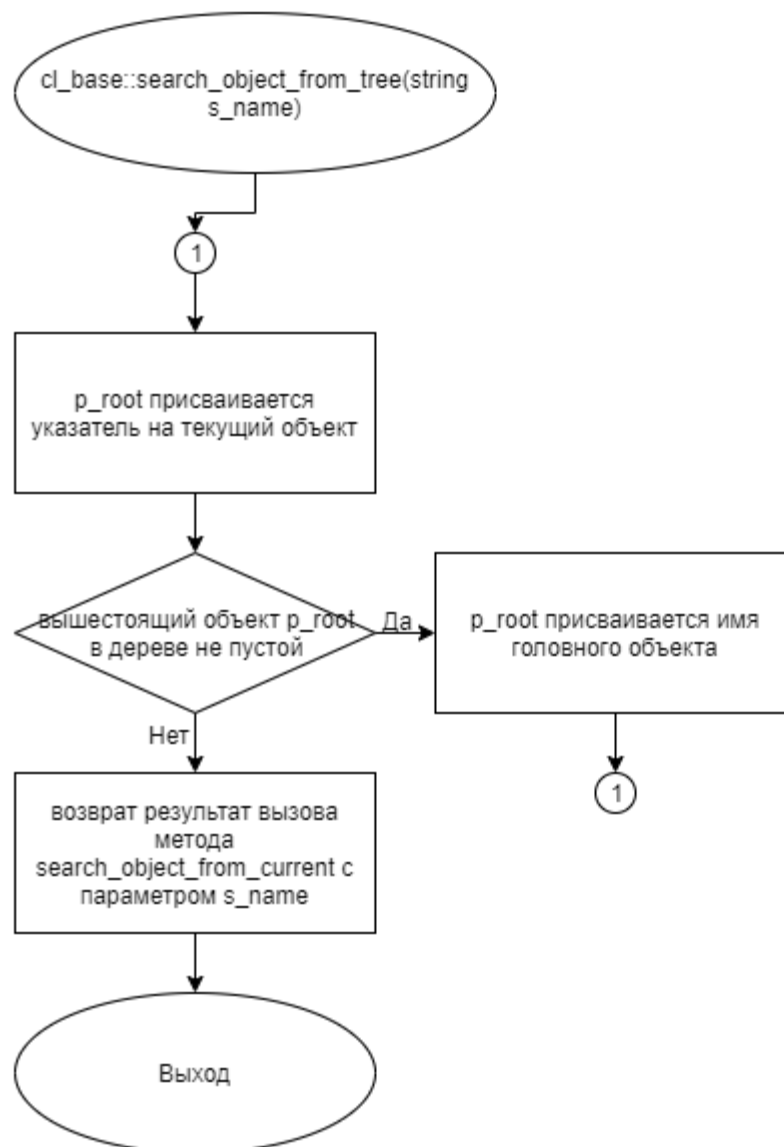


Рисунок 11 – Блок-схема алгоритма

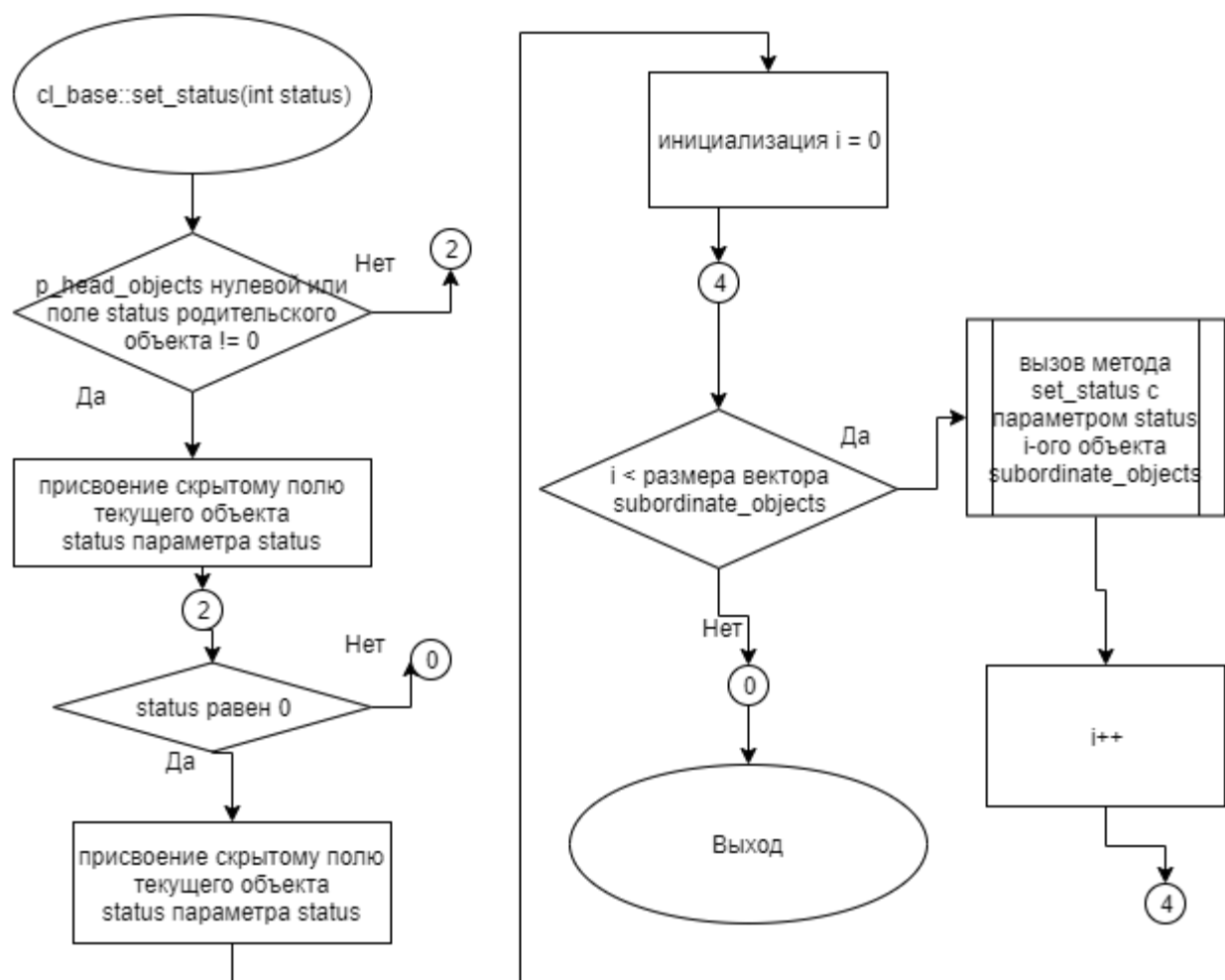


Рисунок 12 – Блок-схема алгоритма

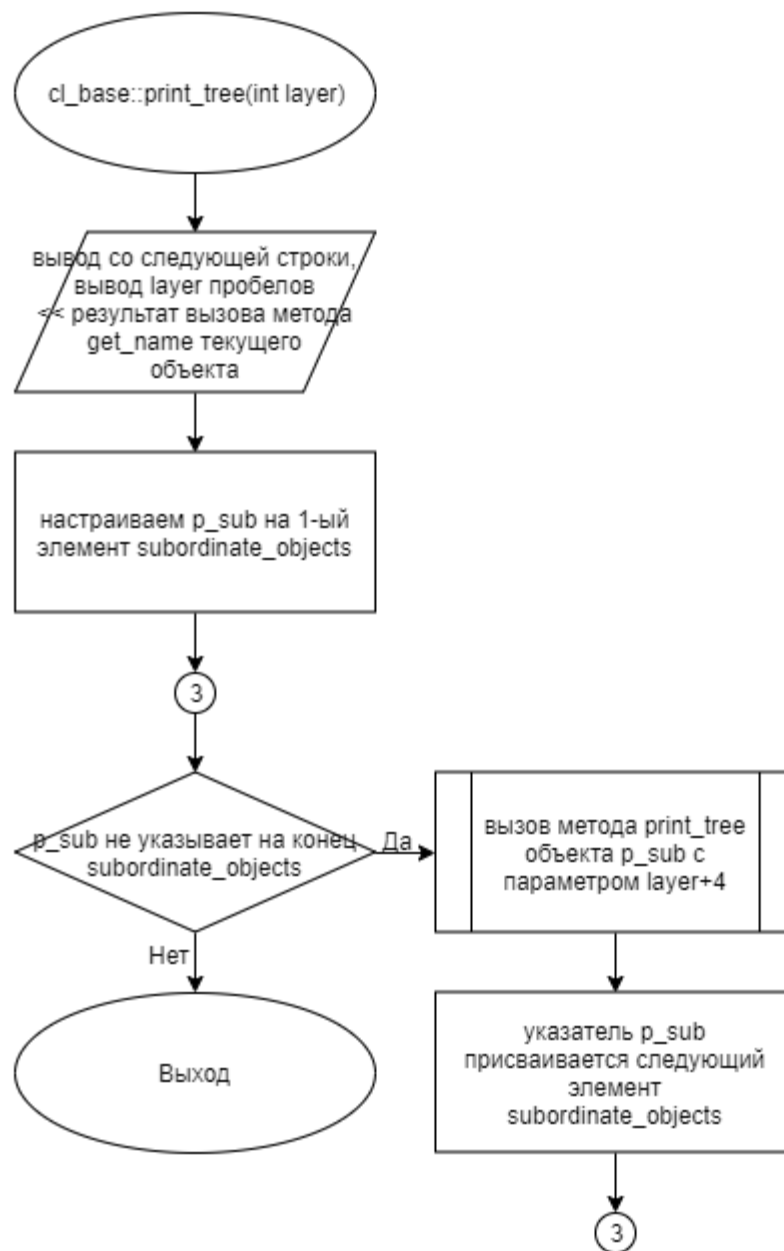


Рисунок 13 – Блок-схема алгоритма

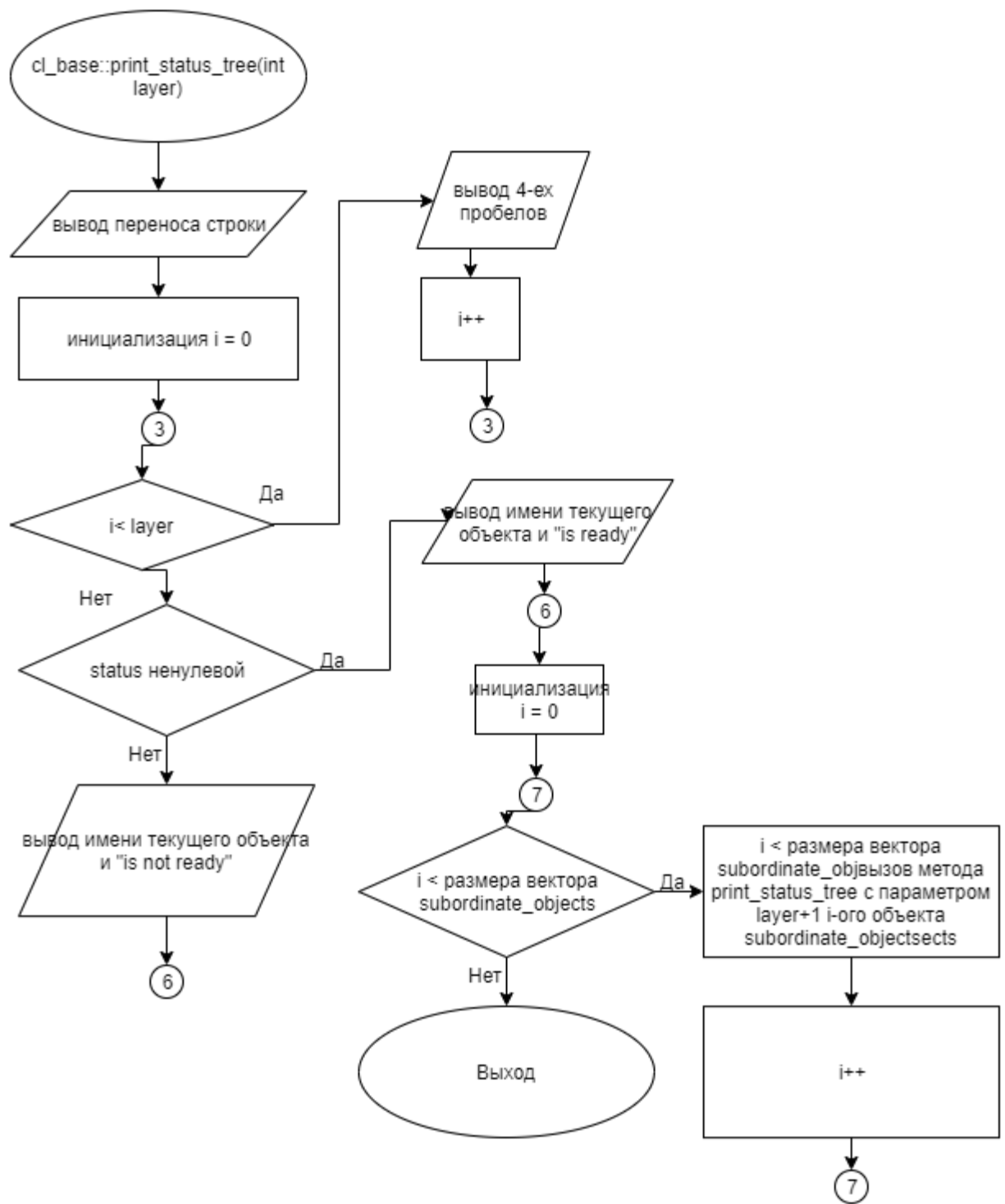


Рисунок 14 – Блок-схема алгоритма

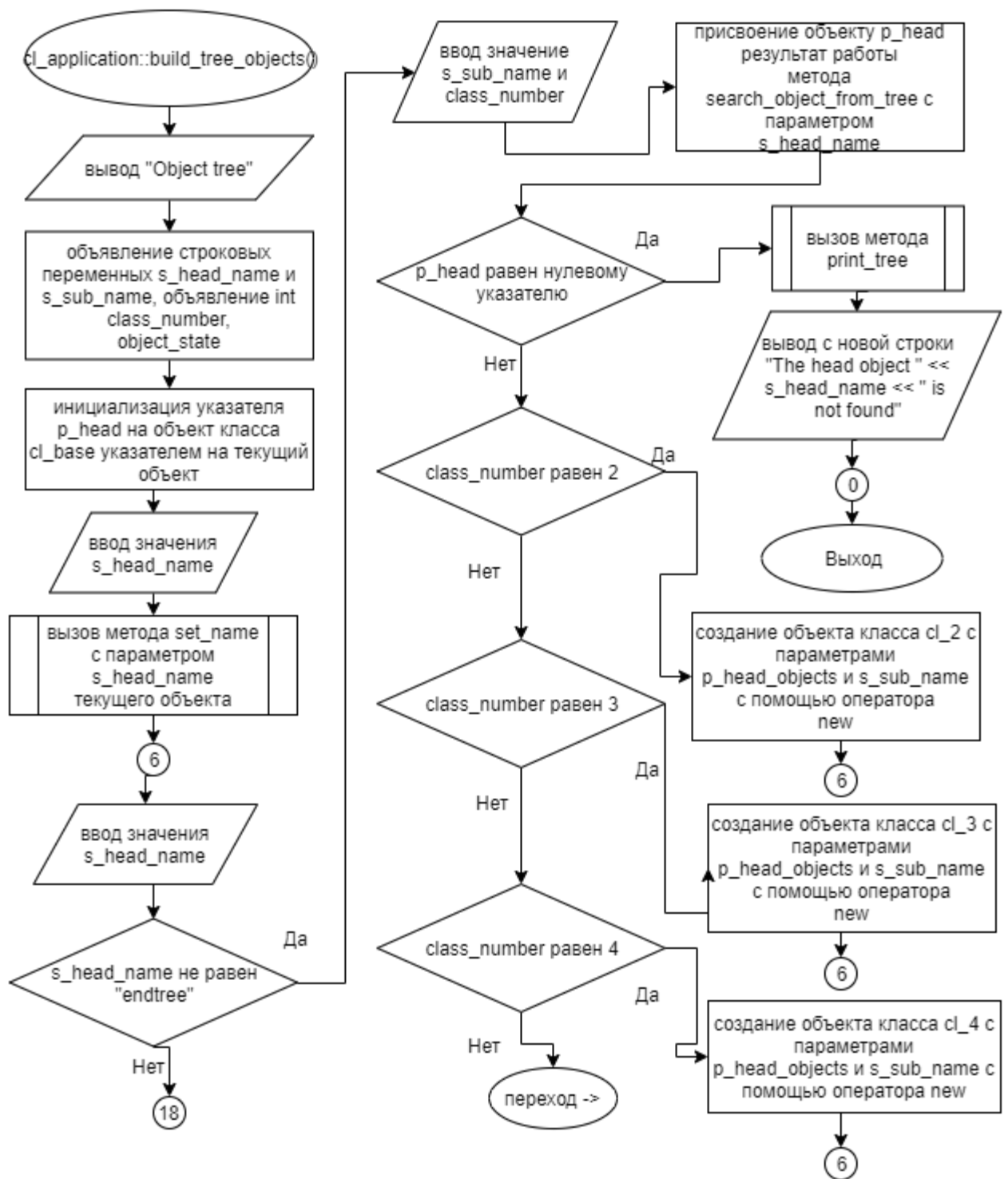


Рисунок 15 – Блок-схема алгоритма

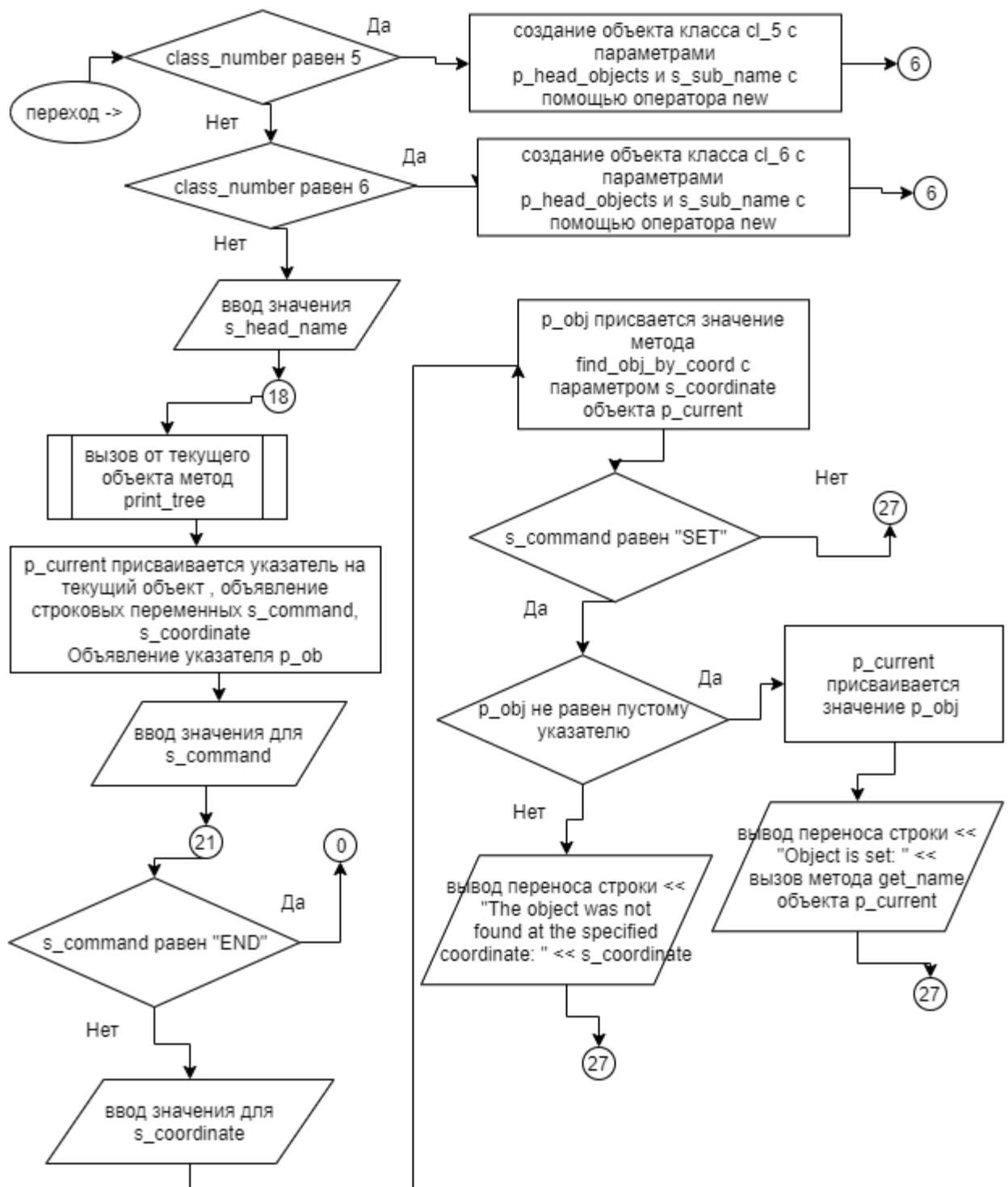


Рисунок 16 – Блок-схема алгоритма

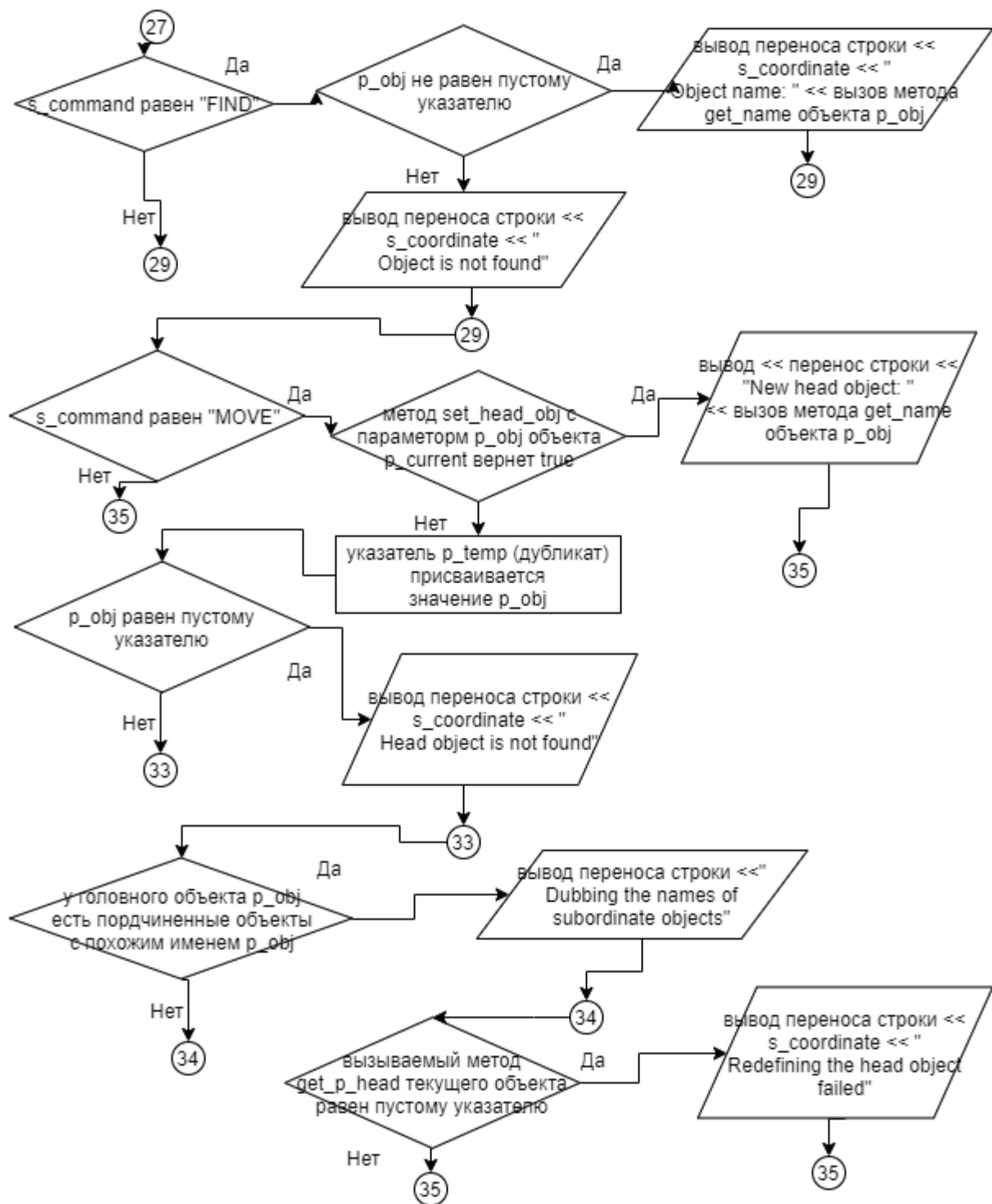


Рисунок 17 – Блок-схема алгоритма

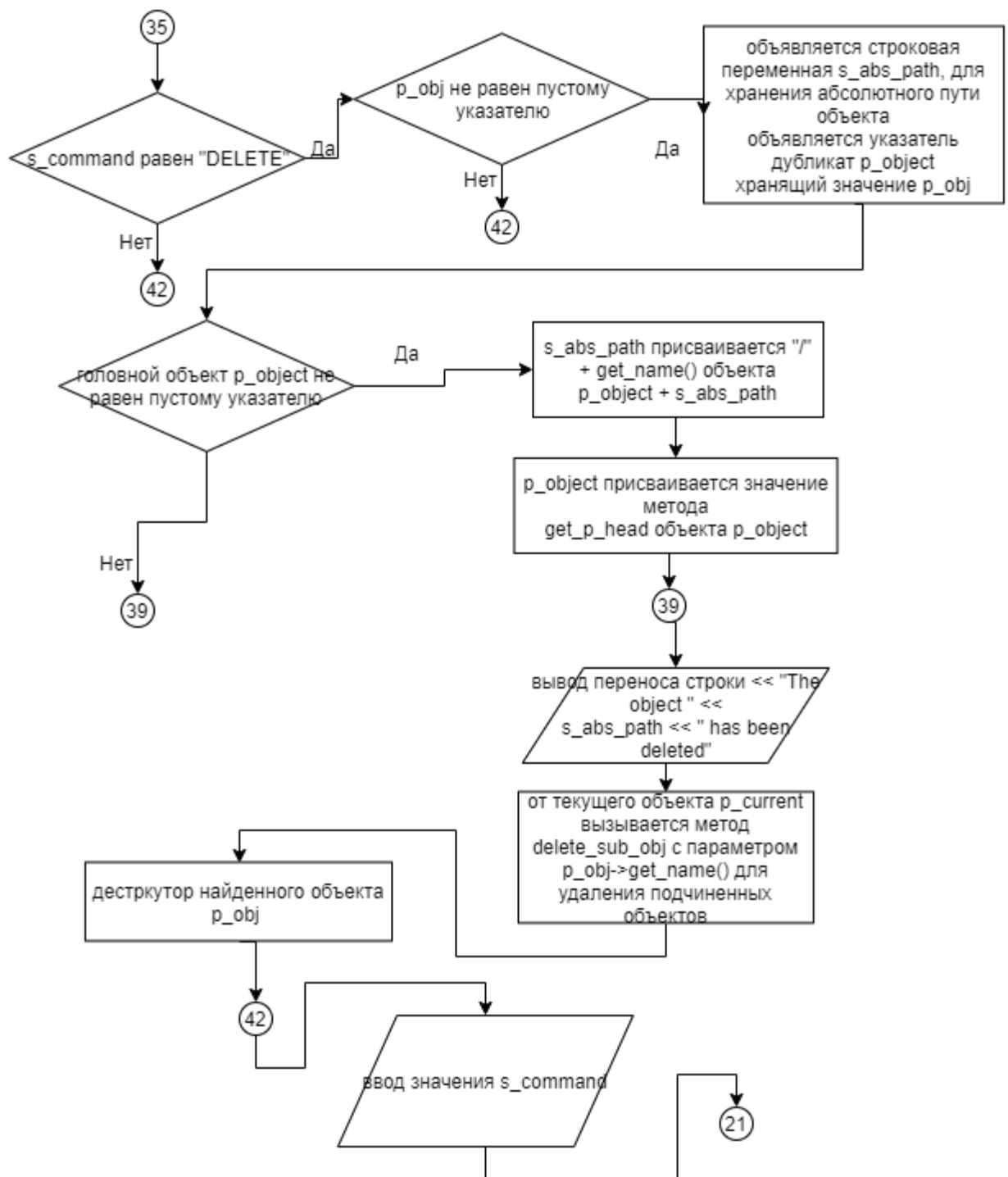


Рисунок 18 – Блок-схема алгоритма

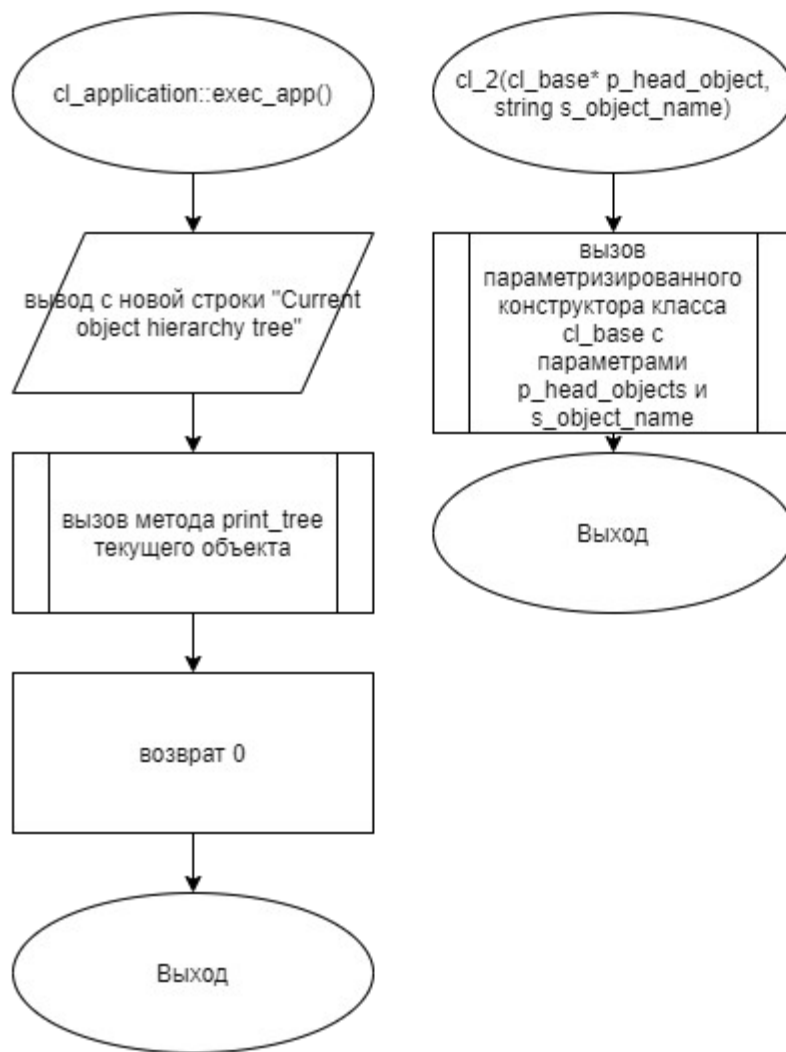


Рисунок 19 – Блок-схема алгоритма

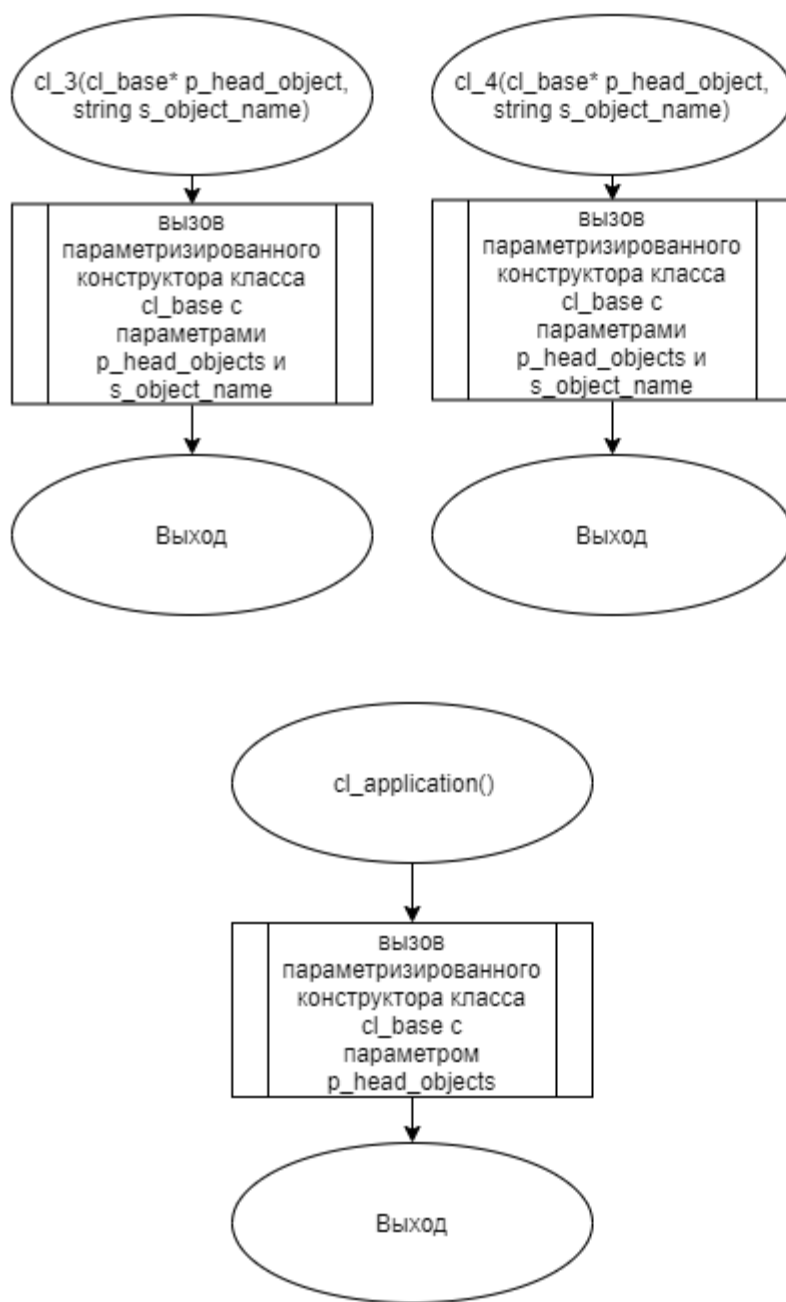


Рисунок 20 – Блок-схема алгоритма

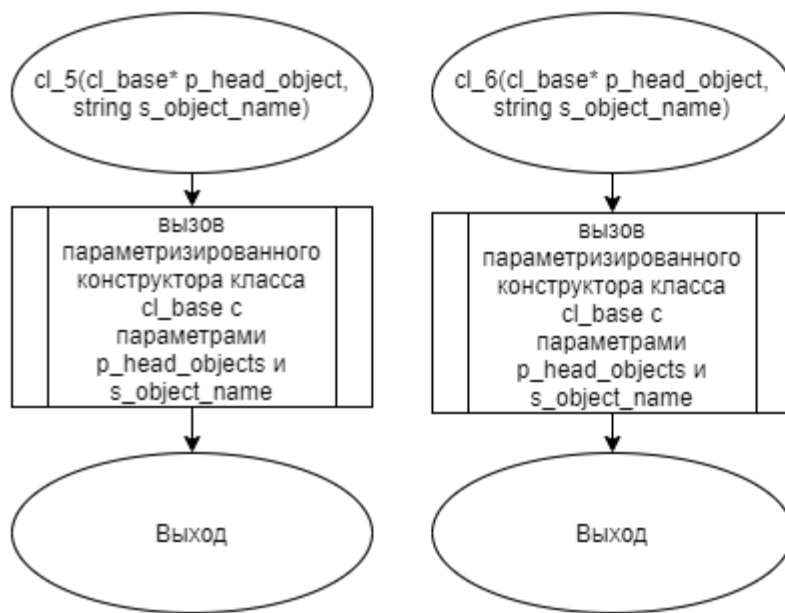


Рисунок 21 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"
//вызов параметризованного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_2::cl_2(cl_base*p_head_object,                                string
s_object_name):cl_base(p_head_object, s_object_name){}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"
class cl_2:public cl_base
{
public: //конструктор, создающий объект класса cl_2
cl_2(cl_base*p_head_object, string s_object_name);
};
#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
//вызов параметризованного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_3::cl_3(cl_base*p_head_object,                                string
s_object_name):cl_base(p_head_object, s_object_name){}
```


5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"
class cl_3:public cl_base
{
public: //конструктор, создающий объект класса cl_3
cl_3(cl_base*p_head_object, string s_object_name);
};
#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"
//вызов параметризованного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_4::cl_4(cl_base*p_head_object, string
s_object_name):cl_base(p_head_object, s_object_name){}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include "cl_base.h"
class cl_4:public cl_base
{
public: //конструктор, создающий объект класса cl_4
cl_4(cl_base*p_head_object, string s_object_name);
};
#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"
//вызов параметризованного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_5::cl_5(cl_base*p_head_object,                                string
s_object_name):cl_base(p_head_object, s_object_name){}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include "cl_base.h"
class cl_5:public cl_base
{
public: //конструктор, создающий объект класса cl_5
cl_5(cl_base*p_head_object, string s_object_name);
};
#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"
//вызов параметризованного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_6::cl_6(cl_base*p_head_object,                                string
s_object_name):cl_base(p_head_object, s_object_name){}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "cl_base.h"
class cl_6:public cl_base
{
public: //конструктор, создающий объект класса cl_6
cl_6(cl_base*p_head_object, string s_object_name);
};
#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
//вызов параметризованного конструктора класса cl_base с параметром
p_head_object
cl_application::cl_application(cl_base*p_head_object):cl_base(p_head_object)
{}
void cl_application::build_tree_objects()
{
    /*
    метод построения дерева иерархии объектов
    */
    cout << "Object tree";

    string s_head_name, s_sub_name;
    cl_base *p_head;
    cl_base *p_sub = nullptr;
    int class_number, object_state;
    cin >> s_head_name;
    // вызов метода set_name этого объекта с параметром s_head_name
    this->set_name(s_head_name);
    // присваиваем p_head этот объект
    p_head = this;
    cin >> s_head_name;
    // ввод иерархии объектов
    while(s_head_name != "endtree"){
        cin >> s_sub_name >> class_number;
        //поиск головного объекта
        p_head = find_obj_by_coord(s_head_name); // кв3
        // если головной объект ненулевой
        if (p_head == nullptr){
            print_tree();
            cout << endl << "The head object " << s_head_name << " is not
```

```

found";
    exit(1);
}
else{
    // далее - существует единственный головной объект p_head и
    подчиненного с таким именем не существует
    switch(class_number){
        case 2:
            p_sub = new cl_2(p_head, s_sub_name);
            break;
        case 3:
            p_sub = new cl_3(p_head, s_sub_name);
            break;
        case 4:
            p_sub = new cl_4(p_head, s_sub_name);
            break;
        case 5:
            p_sub = new cl_5(p_head, s_sub_name);
            break;
        case 6:
            p_sub = new cl_6(p_head, s_sub_name);
            break;
    }
}
cin >> s_head_name;
}
//вызов метода print_tree этого объекта
this->print_tree();
string s_command, s_coordinate; // команда, координата
cl_base* p_current = this; // указатель на текущий объект (головной объект
класса cl_application)
cl_base* p_obj;
cin >> s_command;
while(s_command != "END"){
    cin >> s_coordinate;
    p_obj = p_current->find_obj_by_coord(s_coordinate); // ищем переданный
    объект по координатам
    if(s_command == "SET"){
        // если объект явл. ненулевым указателем (т.е. найден)
        if (p_obj != nullptr){
            p_current = p_obj; // текущему объекту присваиваем найденный
            объект
            cout << endl << "Object is set: " << p_current->get_name();
        }
        else{
            cout << endl << "The object was not found at the specified
            coordinate: " << s_coordinate;
        }
    }
    else if(s_command == "FIND"){
        // если объект явл. ненулевым указателем (т.е. найден)
        if (p_obj != nullptr){
            cout << endl << s_coordinate << "      Object name: " << p_obj-
            >get_name();
        }
    }
}

```

```

        else{
            cout << endl << s_coordinate << "      Object is not found";
        }
    }
    // смена головного объекта
    else if(s_command == "MOVE"){
        // если метод, вызванный через текущий объект с параметром
        найденного объекта вернет true
        if (p_current->set_head_obj(p_obj)){
            cout << endl << "New head object: " << p_obj->get_name();
        }
        else{
            // если головной элемент равен нулевому указателю
            if (p_obj == nullptr){
                cout << endl << s_coordinate << "      Head object is not
found";
            }
            // если у нового головного объекта есть подчиненные с одним
            именем
            if (p_obj-> get_subordinate_object(p_current-> get_name()) !=
            nullptr){
                cout << endl << s_coordinate << "      Dubbing the names of
subordinate objects";
            }
            // если корневой элемент равен нулевому указателю
            else if (this->get_p_head() == nullptr){
                cout << endl << s_coordinate << "      Redefining the head
object failed";
            }
        }
    }
    else if(s_command == "DELETE"){
        // если p_obj не равен пустому указателю, то мы нашли объект,
        который нужно удалить => он существует
        if (p_obj != nullptr){
            string s_abs_path = ""; // абсолютный путь объекта
            cl_base* p_object = p_obj;
            // поднимаемся по дереву, пока головной объект не равен пустому
            указателю
            while (p_object -> get_p_head() != nullptr){
                s_abs_path = "/" + p_object->get_name() + s_abs_path; //
составляем путь объекта
                p_object = p_object->get_p_head();
            }
            cout << endl << "The object " << s_abs_path << " has been
deleted";
            // p_obj->get_name() s_coordinate
            p_current-> delete_sub_obj(p_obj->get_name()); // от текущего
            объекта удаляется подчиненные объекты p_object->get_name()
            delete p_obj; // деструктор найденного объекта
        }
    }
    cin >> s_command;
}
}
}

```

```

int cl_application::exec_app()
{
    cout << endl << "Current object hierarchy tree";
    // вызов метода print_tree этого объекта
    this->print_tree();
    return 0;
}

```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```

#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.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);    //конструктор, создающий объект
    класса cl_app
    void build_tree_objects(); //метод, создающий иерархию объекта
    int exec_app(); //метод запуска приложения
};
#endif

```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```

#include "cl_base.h"
cl_base::cl_base(cl_base*p_head_object,                                string
s_object_name):p_head_object(p_head_object), s_object_name(s_object_name)
{
    /*
    параметризованный конструктор
    p_head_object - указатель на головной объект
    s_object_name - имя узла дерева
    */
    //полю p_head_object этого объекта присваивается параметр p_head_object
}

```

```

        //полю s_object_name этого объекта присваивается параметр s_object_name

        //если p_head_object ненулевой, то в subordinate_objects добавляется
        указатель на этот объект
        if(p_head_object)
        {
            p_head_object->subordinate_objects.push_back(this);
        }
    }
    cl_base::~~cl_base()
    {
        //проходимся по каждому элементу subordinate_objects и удаляем его
        for(auto p_sub: subordinate_objects)
        {
            delete p_sub;
        }
    }

    bool cl_base::set_name(string new_name)
    {
        /*
        метод редактирования имени объекта
        new_name - новое имя узла дерева
        */
        //проходимс по каждому эдементу вектора указателей subordinate_objects
        объекта по указателю p_head_objects
        //если он равен new_name, то возвращаем false
        if (p_head_object != nullptr)
        {
            for (auto p_sub : p_head_object->subordinate_objects)
            {
                if (p_sub->get_name() ==new_name)
                {
                    return false;
                }
            }
        }
        // полю s_object_name этого объекта присваивается new_name
        this->s_object_name = new_name;
        return true;
    }

    string cl_base::get_name()
    {
        //возвращаем s_object_name
        return this->s_object_name;
    }

    cl_base * cl_base::get_p_head()
    {
        //возвращаем p_head_object
        return this->p_head_object;
    }

    cl_base * cl_base::get_subordinate_object(string search_name)

```

```

{
    /*
    получение указателя на непосредственно подчиненный объект по имени
    search_name - имя искомого объекта
    */

    //проходимся по элементам subordinate_objects, если он равен search_name
    //возвращаем i-ым subordinate_objects
    for(auto p_sub: subordinate_objects)
    {
        if(p_sub -> get_name() == search_name)
        {
            return p_sub;
        }
    }
    //иначе возвращается нулевой указатель
    return nullptr;
}

cl_base* cl_base::search_object_from_current(string s_name)
{
    /*
    метод поиска объекта по имени в поддереве(в ветке) (обход графа в ширину)
    s_name - имя искомого объекта
    */

    cl_base* p_found = nullptr; //указатель на объект, который был найден
    queue<cl_base*> q; //очередь элемента

    q.push(this); //добавляем в очередь текущий элемент

    //пока очередь не пустая
    while(!q.empty())
    {
        cl_base* p_front = q.front(); //хранится указатель на наш объект
        q.pop(); //удаляем элемент из начала очереди, чтобы пройти по всем
        элементам
        if (p_front->get_name() == s_name) //если имя указателя на элемент в
        очереди совпадает с искомым объектом
        {
            if(p_found == nullptr) //указатель на пустой и объект не уникальный
            p_found = p_front; //присваиваем указатель на элемент в очереди
            else //нашли дубликат
            return nullptr;
        }
        // добавляем дочерние элементы p_front в очередь
        for(auto p_sub : p_front->subordinate_objects)
        {
            q.push(p_sub);
        }
    }
    return p_found;
}

cl_base* cl_base::search_object_from_tree(string s_name)

```



```

{
    /*
    поиск объекта по имени во всем дереве
    s_name - имя искомого объекта
    */

    cl_base* p_root = this;
    //пока вышестоящий объект в дереве не пустой, поднимаемся по дереву
    while(p_root -> get_p_head() != nullptr)
    {
        p_root = p_root -> get_p_head();
    }
    return p_root->search_object_from_current(s_name);
}

void cl_base::print_tree(int layer)
{
    /*
    метод вывода иерархии объектов(дерева/ветки) от текущего объекта
    layer - уровень на дереве иерархии
    */

    //выводим layer-ое кол-во ' ' и имя объекта
    cout << endl << string(layer, ' ') << this -> get_name();
    //проходимся по элементам subordinate_objects и вызываем рекурсию
    for(auto p_sub:subordinate_objects)
        p_sub->print_tree(layer + 4);
}

void cl_base::print_status_tree(int layer)
{
    /*
    метод вывода дерева/ветки иерархии объектов и их статуса от текущего
    объекта
    layer - уровень на дереве иерархии
    */

    cout << endl;
    for(int i=0; i<layer; ++i)
    {
        cout << "    ";
    }
    //проверка на статус текущего объекта
    if(this -> status != 0)
    {
        cout << this -> get_name() << " is ready";
    }
    else
    {
        cout << this ->get_name() << " is not ready";
    }
    for(int i=0; i<subordinate_objects.size(); ++i)
    {
        subordinate_objects[i] -> print_status_tree(layer + 1);
    }
}

```

```

}

void cl_base::set_status(int status)
{
    /*
    метод установки статуса объекта
    status - номер состояния
    */
    //если значение status ненулевое у головного объекта
    if(p_head_object == nullptr || p_head_object -> status != 0)
    {
        this -> status = status;
    }
    if(status == 0)
    {
        this -> status = status;
        for(int i=0; i<subordinate_objects.size(); i++)
        {
            subordinate_objects[i] -> set_status(status);
        }
    }
}

cl_base* cl_base::find_obj_by_coord(string s_coord)
{
    /*
    метод поиска объекта по координате
    s_coord - координата искомого объекта
    */

    cl_base* p_root = this; // указатель на текущий объект
    int i_slash_2 = 0; // хранит индекс 2-ого слэша
    string s_name = "";
    cl_base* p_obj;
    if (s_coord == "/"){
        //поднимаемся по корню дереву
        while(p_root -> get_p_head() != nullptr){ // пока головной объект не
        равен пустому указателю
            p_root = p_root -> get_p_head();
        }
        return p_root; // нашли корневой объект
    }
    if (s_coord == "."){
        return this; // возвращаем текущий объект, т.к. считаем, что метод
        вызван от текущего объекта
    }
    if (s_coord[0] == '/' && s_coord[1] == '/'){ // второй символ строки равен
    (/ '/')
        return this->search_object_from_tree(s_coord.substr(2)); // текущему
        объекту вызываем метод с парам. s_coord (начиная с 3 символа)
    }
    if (s_coord[0] == '.'){
        return this->search_object_from_current(s_coord.substr(1)); // текущему
        объекту вызываем метод с парам. s_coord (начиная со 2 символа)
    }
}

```

```

i_slash_2 = s_coord.find("/",1);
if (s_coord[0] == '/') {
    //поднимаемся по корню дерева
    while(p_root -> get_p_head() != nullptr){ // пока головной объект не
        равен пустому указателю
        p_root = p_root -> get_p_head();
    }
    if (i_slash_2 != -1){ // нашли индекс слэша
        s_name = s_coord.substr(1, i_slash_2 - 1); // получили имя объекта,
        после слэша и до след. слэша
        p_obj = p_root -> get_subordinate_object(s_name); //
        корневому объекту вызываем метод поиска подчиненных объектов с парам. s_name
        if (p_obj != nullptr) // если p_obj не равен нулевому указателю
            return p_obj->find_obj_by_coord(s_coord.substr(i_slash_2 + 1));
        else
            return p_obj;
    }
    else{
        s_name = s_coord.substr(1);
        return p_root->get_subordinate_object(s_name); // от
        корневого объекта ищем подчиненные
    }
} // ob1/ob2/ob3
else{
    if (i_slash_2 != -1){ // если слэш был найден в s_coord
        s_name = s_coord.substr(0, i_slash_2); // получили имя объекта
        p_obj = this -> get_subordinate_object(s_name); // текущему
        объекту вызываем метод с парам. s_name
        if (p_obj != nullptr) // если p_obj не равен нулевому указателю
            return p_obj->find_obj_by_coord(s_coord.substr(i_slash_2
+1)); //рекурсивно вызываем этот метод, передавая строку, после / в s_coord
        else
            return p_obj;
    }
    else{ // второго слэша не нашлось (ob3)
        return this->get_subordinate_object(s_coord); // от текущего
        объекта ищем подчиненные
    }
}
return nullptr;
}
void cl_base::delete_sub_obj(string s_name){
    /*
    метод удаления подчиненного объекта по имени
    s_name - имя удаляемого объекта
    */
    // проходимся по подчиненным элементам
    for (int i = subordinate_objects.size() - 1; i>=0; --i){
        if (subordinate_objects[i]-> get_name() == s_name){ // если подчиненный
        элемент равен имени удаляемого объекта
            subordinate_objects.erase(subordinate_objects.begin() + i);
            break;
        }
    }
}

```

```

    }
}
bool cl_base::set_head_obj(cl_base* p_new_head){
    /*
    метод смены головного элемента
    p_new_head - имя нового головного объекта
    */
    cl_base* p_temp = p_new_head; // указатель
    // если головной элемент равен нулевому указателю
    if (p_new_head == nullptr){
        return false;
    }
    // если корневой элемент равен нулевому указателю
    if (this->get_p_head() == nullptr){
        return false;
    }
    // если у нового головного объекта есть подчиненные с одним именем
    if (p_new_head-> get_subordinate_object(this-> get_name()) != nullptr){
        return false;
    }
    // поднимаемся по дереву от нашего нового головного объекта, пока не найдем
    до корня
    while (p_temp != nullptr){
        if (p_temp == this){ // если p_temp равен текущему объекту
            (является подчиненным - недопустимо)
            return false;
        }
        else
            p_temp = p_temp -> get_p_head();
    }
    // головной объект удалили из подчиненных у текущего головного объекта
    this -> get_p_head()->delete_sub_obj(this->get_name());
    // присваиваем имя нового головного объекта
    // новому головному объекту добавляем в подчиненные объекты текущий объект
    p_new_head->subordinate_objects.push_back(this);
    return true;
}

```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```

#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <vector>
#include <string>
#include <queue>
using namespace std;
class cl_base

```

```

{
private:
    int status = 0; //статус состояния объектов
    string s_object_name; //имя объекта
    cl_base*p_head_object; //указатель на родительский объект
    vector <cl_base *> subordinate_objects; //вектор подчиненных объектов
public:
    cl_base(cl_base*p_head_object,    string    s_object_name="Base    object");
//параметризированный конструктор
    ~cl_base(); //деструктор
    bool set_name(string new_name); //метод редактирования имени объекта
    string get_name(); //метод получения имени
    cl_base*get_p_head(); //метод получения указателя на родительский объект
    cl_base*get_subordinate_object(string search_name); //получение указателя
на непосредственно подчиненный объект по имени
    cl_base*search_object_from_current(string); //поиск объекта на ветке
иерархии от текущего имени
    cl_base*search_object_from_tree(string); //поиск по дереву(в корне)
иерархии
    void print_tree(int layer=0); //метод вывода дерева иерархии
    void print_status_tree(int layer=0); //метод вывода дерева/ветки иерархии
объектов и их статуса от текущего объекта
    void set_status(int status); //метод установки статуса объекта
    cl_base* find_obj_by_coord(string); //метод поиска объекта по заданной
координате
    void delete_sub_obj(string); //метод удаления подчиненного объекта по
имени
    bool set_head_obj(cl_base*); //метод смены головного элемента
};
#endif

```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```

#include "cl_application.h"
int main()
{
    cl_application ob_cl_application(nullptr); //создание корневого объекта
    ob_cl_application.build_tree_objects(); //конструирование системы,
построение дерева иерархии
    return (ob_cl_application.exec_app()); //запуск системы
}

```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 25.

Таблица 25 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
<pre> rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 SET /object_2 FIND //object_7 FIND object_4/object_7 FIND . FIND .object_7 FIND object_4/object_7 MOVE .object_7 SET object_4/object_7 MOVE //object_1 MOVE /object_3 END </pre>	<pre> Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_7 Object is not found object_4/object_7 Object name: object_7 . Object name: object_2 .object_7 Object name: object_7 object_4/object_7 Object name: object_7 .object_7 Redefining the head object failed Object is set: object_7 //object_1 Dubbing the names of subordinate objects New head object: object_3 Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_5 </pre>	<pre> Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_7 Object is not found object_4/object_7 Object name: object_7 . Object name: object_2 .object_7 Object name: object_7 object_4/object_7 Object name: object_7 .object_7 Redefining the head object failed Object is set: object_7 //object_1 Dubbing the names of subordinate objects New head object: object_3 Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_5 </pre>

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	<pre> object_3 object_3 object_7 </pre>	<pre> object_3 object_3 object_7 </pre>
<pre> rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree DELETE /object_2 END </pre>	<pre> Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 The object /object_2 has been deleted Current object hierarchy tree rootela object_1 object_7 object_3 </pre>	<pre> Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 The object /object_2 has been deleted Current object hierarchy tree rootela object_1 object_7 object_3 </pre>
<pre> rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 SET /object_2 FIND //object_7 FIND object_4/object_7 FIND . FIND .object_7 FIND object_4/object_7 FIND /.//. FIND / FIND .object_7 FIND object_4/object_7 MOVE .object_7 SET object_4/object_7 </pre>	<pre> Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_7 Object is not found object_4/object_7 Object name: object_7 . Object name: object_2 .object_7 Object name: object_7 object_4/object_7 Object name: object_7 /.//. Object is not found / Object name: </pre>	<pre> Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_7 Object is not found object_4/object_7 Object name: object_7 . Object name: object_2 .object_7 Object name: object_7 object_4/object_7 Object name: object_7 /.//. Object is not found / Object name: </pre>

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
<pre> MOVE //object_1 MOVE /object_3 SET /object_2 DELETE object_5 DELETE /object_7 END </pre>	<pre> rootela .object_7 Object name: object_7 object_4/object_7 Object name: object_7 .object_7 Redefining the head object failed Object is set: object_7 //object_1 Dubbing the names of subordinate objects New head object: object_3 Object is set: object_2 The object /object_2/object_5 has been deleted Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_3 object_3 object_7 </pre>	<pre> rootela .object_7 Object name: object_7 object_4/object_7 Object name: object_7 .object_7 Redefining the head object failed Object is set: object_7 //object_1 Dubbing the names of subordinate objects New head object: object_3 Object is set: object_2 The object /object_2/object_5 has been deleted Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_3 object_3 object_7 </pre>

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-avvora.ru/student/files/methodichescoc_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-avvora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).
4. Шилдт Г. С++: 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).