Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

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:
    - / - корневой объект;
    - //«имя объекта» - поиск объекта по уникальной имени от корневого (для однозначности уникальность требуется в рамках дерева);
    - . - текущий объект;
    - .«имя объекта» - поиск объекта по уникальной имени от текущего (для однозначности уникальность требуется в рамках ветви дерева от

текущего объекта);

- «имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;
- /«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 (дубляж 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 отрабатывает 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 головному 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

Для решения задачи используется:

- объект cin класса потокового ввода предназначен для функционирования системы;
- объект cout класса потокового вывода предназначен для функционирования системы;
- объект obj_custom_app класса cl_custom_application предназначен для запуска приложения;
- объект класса cl_custom_2, cl_custom_3, cl_custom_4, cl_custom_5, cl_custom_6 предназначен для формирования иерархической структуры в дереве объектов;
- объект p_sub_customs класса cl_custom_base предназначен для хранение подчиненных объектов;
- оператор new - выделение динамической памяти для объектов;
- оператор return - возврат значения из функции;
- оператор delete - освобождение памяти.

Класс cl_custom_base:

- свойства/поля:
    - поле , строка, представляющая имя пользовательского объекта:
        - наименование — s_name;
        - тип — строковый;
        - модификатор доступа — private;
    - поле , указатель, на объект родитель текущего пользовательского объекта:
        - наименование — p_main_custom_object;
        - тип — cl_custom_base;

        - модификатор доступа — private;
    - поле , хранение состояния объекта:
        - наименование — i_state;
        - тип — целочисленный;
        - модификатор доступа — private;
- функционал:
    - метод cl_custom_base — конструктор;
    - метод ~cl_custom_base — деструктор;
    - метод get_custom_name — получние имени текущего пользовательского объекта, возврат строки, представляющую имя объекта;
    - метод get_main_custom — получение указателя на подпользовательский объект текущего пользовательского объекта;
    - метод get_sub_custom — получение указателя на подпользовательский объект текущего пользовательского объекта;
    - метод search_tree — поиск объекта в дереве;
    - метод search_current — поиск объекта начиная от текущего объекта;
    - метод print_custom_tree — вывод информации о пользовательских объектах и их иерархии на экран;
    - метод print_tree_two — вывод информации о пользовательских объектах и их иерархии на экран;
    - метод set_head_object — переопределение головного объекта для текущего в дереве иерархии;
    - метод delete_custom_sub_name — удаление подчиненного объекта по наименованию;
    - метод get_object_path — получение указателя на любой объект в составе дерева иерархии объектов.

Класс cl_custom_application:

- функционал:
    - метод build_tree_objects — создание иерархии объектов;
    - метод exec_custom_app — сборка приложения;
    - метод cl_custom_application — конструктор.

Класс cl_custom_2:

- функционал:
    - метод cl_custom_2 — конструктор.

Класс cl_custom_3:

- функционал:
    - метод cl_custom_3 — конструктор.

Класс cl_custom_4:

- функционал:
    - метод cl_custom_4 — конструктор.

Класс cl_custom_5:

- функционал:
    - метод cl_custom_5 — конструктор.

Класс cl_custom_6:

- функционал:
    - метод cl_custom_6 — конструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_custom_base			основная логика управления пользовательскими объектами и их иерархией	
		cl_custom_application	public		2

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
		cl_custom_2	public		3
		cl_custom_3	public		4
		cl_custom_4	public		5
		cl_custom_5	public		6
		cl_custom_6	public		7
2	cl_custom_application			наследуется от 'cl_custom_base', представляет пользовательское приложение и содержит логику его выполнения	
3	cl_custom_2			создание типа объекта, наследуя от 'cl_custom_base', собственные свойства отсутствуют	
4	cl_custom_3			создание типа объекта, наследуя от 'cl_custom_base', собственные свойства отсутствуют	
5	cl_custom_4			создание типа объекта, наследуя от 'cl_custom_base', собственные свойства отсутствуют	
6	cl_custom_5			создание типа объекта, наследуя от 'cl_custom_base', собственные свойства отсутствуют	
7	cl_custom_6			создание типа объекта, наследуя от 'cl_custom_base', собственные свойства отсутствуют	

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 конструктора класса cl_custom_base

Функционал: конструктор.

Параметры: p_main - указатель на главный объект, s_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация указателя p_main_custom_object текущего объекта значением p_main, которое передано в качестве аргумента	2
2		присвоение значения аргумента s_name члену класса custom_name текущего объекта	3
3	не является ли p_main_custom_object пустым?	добавление объекта в вектор подпользовательских объектов p_sub_customs родительского класса 'p_main_custom_object'	∅
			∅

3.2 Алгоритм деструктора класса cl_custom_base

Функционал: деструктор.

Параметры: отсутствуют.

Алгоритм деструктора представлен в таблице 3.

Таблица 3 – Алгоритм деструктора класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация целочисленной переменной i = 0	2
2	i < размера вектора p_sub_customs	i++; удаление p_sub_customs[i], освобождение памяти	∅
			∅

3.3 Алгоритм метода get_custom_name класса cl_custom_base

Функционал: получние имени текущего пользовательского объекта, возврат строки, представляющую имя объекта.

Параметры: отсутствуют.

Возвращаемое значение: имя текущего объекта типа cl_custom_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_custom_name класса cl_custom_base

№	Предикат	Действия	№ перехода
1		возврат значения поля custom_name текущего объекта	∅

3.4 Алгоритм метода get_main_custom класса cl_custom_base

Функционал: получение указателя на подпользовательский объект текущего пользовательского объекта.

Параметры: отсутствуют.

Возвращаемое значение: значение p_main_custom_object.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_main_custom класса cl_custom_base

№	Предикат	Действия	№ перехода
1		возврат указателя на родительский объект текущего объекта	∅

3.5 Алгоритм метода get_sub_custom класса cl_custom_base

Функционал: получение указателя на подпользовательский объект текущего пользовательского объекта.

Параметры: s_name - имя подчиненного объекта, которое требуется найти среди подчиненных текущего объекта.

Возвращаемое значение: указатель на подчиненный объект, если подчиненного объекта с таким именем нет, возвращается nullptr.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_sub_custom класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация челочисленной переменной i = 0	2
2	i < размера вектора p_sub_customs		3
			4
3	имя текущего подобъекта из списка p_sub_customs равно заданному имени s_name?	возврат указателя на текущий подобъект из списка p_sub_customs , если его имя совпадает с заданным именем s_name	4
			4
4		возврат нулевого указателя	∅

3.6 Алгоритм метода search_tree класса cl_custom_base

Функционал: поиск объекта в дереве.

Параметры: отсутствуют.

Возвращаемое значение: указатель на объект класса cl_custom_base, если объект с таким именем не найден , возврат nullptr.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода search_tree класса cl_custom_base

№	Предикат	Действия	№ перехода
1		объявление очереди с элементами типа указателя на объекты класса cl_custom_base	2
2		инициализация целочисленной переменной i = 0	3
3		объявление указателя found на объект класса cl_custom_base и инициализация его значением nullptr	4
4		добавление указателя this , который указывает на текущий объект класса cl_custom_base, в конец очереди q	5
5	очередь q не пустая	создание временного указателя e, который указывает на первый элемент в очереди q	6
			10
6	найден ли объект с именем , которое соответствует s_name		7
			8
7	был ли найден объект с именем s_name	возврат нулевого указателя	8
		присвоение указателя found значеним e	8
8	i меньше размера вектора p_sub_customs текущего объекта e	добавление текущего подчинненого объекта в очередь q	9
			9

№	Предикат	Действия	№ перехода
9		удаление первого элемента из очереди q	10
10		возврат указателя found	∅

3.7 Алгоритм метода search_current класса cl_custom_base

Функционал: поиск объекта начиная от текущего объекта.

Параметры: s_name - имя объекта, который нужно найти.

Возвращаемое значение: указатель на объект, который имеет имя s_name или возврат nullptr.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода search_current класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация целочисленной переменной i = 0	2
2	главный объект p_main_custom_object не равен nullptr	рекурсивный возврат метода search_current с параметром s_name	∅
		рекурсивный возврат метода search_current с параметром s_name	∅

3.8 Алгоритм метода print_custom_tree класса cl_custom_base

Функционал: вывод информации о пользовательских объектах и их иерархии на экран.

Параметры: layer .

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода print_custom_tree класса cl_custom_base

№	Предикат	Действия	№ перехода
1		переход на следующую строку	2
2		инициализация целочисленной переменной i = 0	3
3	i < layer	i++; вывод на экран четерых пробелов	4
			4
4		вывод имени объекта, полученного с помощью функции get_custom_name	5
5	i меньше размера вектора p_sub_customs	i++; вызов метода print_custom_tree для объекта, находящегося на i-том месте в векторе p_sub_customs с увеличсенным значением layer в качестве аргумента	∅
			∅

3.9 Алгоритм метода print_tree_two класса cl_custom_base

Функционал: вывод информации о пользовательских объектах и их иерархии на экран.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода print_tree_two класса cl_custom_base

№	Предикат	Действия	№ перехода
1		переход на следующую строку	2
2		инициализация целочисленной переменной i = 0	3
3	i < layer	вывод на экран 4 пробелов	3
			4
4	layer не равно 0	вывод имени объекта, вывод строки 'is_ready'	5
		вывод имени объекта, вывод строки 'is_not_ready'	5

№	Предикат	Действия	№ перехода
5	i меньше размера p_sub_customs	рекурсивный вызов метода print_tree_two для каждого подчиненного объекта, увеличение параметра i_space на 1	5
			∅

3.10 Алгоритм метода set_head_object класса cl_custom_base

Функционал: переопределение головного объекта для текущего в дереве иерархии.

Параметры: new_p_head_object - указатель на объект типа cl_custom_base.

Возвращаемое значение: true/false.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода set_head_object класса cl_custom_base

№	Предикат	Действия	№ перехода
1	текущий объект является головным объектом	возврат true	2
			2
2	головной объект текущего объекта или переданный объект равны nullptr?	возврат false	3
			3
3	существует ли у нового головного объекта подчиненный объект с тем же именем что и текущий объект	возврат false	4
			4

№	Предикат	Действия	№ перехода
4		создание стека объектов указателей на 'cl_custom_base'	5
5		добавление текущего объекта в стек 'st', начиная	6
6	пуст ли стек st	создание указателя current , который указывает на вершину стека st	7
			11
7		инициализация целочисленной переменной i = 0	8
8	является ли текущий объект равным новому головному объекту	возврат false	9
			9
9	i < размера вектора подчиненных объектов	i++; добавление каждого подчиненного объекта текущего объекта в стек st для дальнейшей обработки	9
			10
10		создание ссылки v, которая указывает на вектор подчиненных объектов главного(родительского) объекта текущего объекта	11
11	i меньше размера v		12
			15
12	имя текущего объекта равноа имени объекта, хранящегося в элементе вектрора v с индексом i	удаление элемента вектора v с индексом i	13
			15
13		добавление текущего объекта в качестве подчиненного объекта нового головного объекта new_p_head_object	14
14		возврат true	15

№	Предикат	Действия	№ перехода
15		возврат false	10

3.11 Алгоритм метода delete_custom_sub_name класса cl_custom_base

Функционал: удаление подчиненного объекта по наименованию.

Параметры: sub_name - наименование подчиненного объекта.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода delete_custom_sub_name класса cl_custom_base

№	Предикат	Действия	№ перехода
1		создание ссылки v, которая указывает на вектор подчиненных объектов текущего объекта	2
2		инициализация целочисленной переменной i	3
3	i < размера v		4
			∅
4	совпадает ли имя подчиненного объекта с заданными наименованием sub_name	удаление объекта, на который указывает элемент вектора с индексом i	5
			∅
5		удаление элемента из вектора v по указанному индексу i	6
6		возврат	∅

3.12 Алгоритм метода get_object_path класса cl_custom_base

Функционал: получение указателя на любой объект в составе дерева

иерархии объектов.

Параметры: path_custom.

Возвращаемое значение: указатель на объект cl_custom_base.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода get_object_path класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация целочисленной переменной i = 0	2
2	пуста ли строка path_custom	возврат нулевого указателя	3
			3
3	является ли строка path_custom точкой "."	возврат указателя на текущий объект cl_custom_base	4
			4
4	является ли первый символ строки path_custom точкой	вызов функции search_tree с аргументом, который является подстрокой path_custom, начиная с индекса 1	5
			5
5	начинается ли строка path_custom с подстроки '//'	вызов метода search_current, передавая ему подстроку path_custom, начиная с позиции 2	6
			6
6	начинается ли строка path_custom с символа '/'	поиск первого вхождения символа '/' в строке path_customs и возврат его позиции	7
			10
7		создание подстроки path_custom, начиная с индекса 0 и имеющую длину slash, вызов метода get_sub_custom, чтобы получить указатель на объект, который соответствует этой подстроке	8
8	подчиненный объект с именем равен подстроке и символ '/' не был найден в	возврат sub_ptr	9

№	Предикат	Действия	№ перехода
	строки path_custom		
			9
9		рекурсивный вызов метода 'get_object_path' у объекта, на который указывает sub_ptr, передавая ему оставшуюся часть пути, начиная со следующего символа после найденного слэша	10
10		присвоение указателя this переменной root, чтобы затем использовать root для итерации до корневого объекта в дереве	11
11	является ли root корневым объектом	i++; обновление root так, чтобы он указывал на родительский объект текущего root	11
			12
12	является ли path_custom корневым путем "/"	возврат указателя на корневой объект	13
			13
13		рекурсивный возврат get_object_path на корневом объекте с путем, начинающего с первого символа после начального '/'	∅

3.13 Алгоритм конструктора класса cl_custom_2

Функционал: конструктор.

Параметры: p_main - указатель на главный объект, s_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса cl_custom_2

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_2', который вызывает	∅

№	Предикат	Действия	№ перехода
		конструктор базового класса 'cl_custom_base' с передачей ему указателя на главный объект 'p_main' и имя объекта 's_name'	

3.14 Алгоритм конструктора класса cl_custom_3

Функционал: конструктор.

Параметры: p_main - указатель на главный объект, s_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 15.

Таблица 15 – Алгоритм конструктора класса cl_custom_3

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_3', который вызывает конструктор базового класса 'cl_custom_base' с передачей ему указателя на главный объект 'p_main' и имя объекта 's_name'	∅

3.15 Алгоритм конструктора класса cl_custom_4

Функционал: конструктор.

Параметры: p_main - указатель на главный объект, s_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 16.

Таблица 16 – Алгоритм конструктора класса cl_custom_4

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_4', который вызывает конструктор базового класса 'cl_custom_base' с передачей ему указателя на главный объект 'p_main' и имя объекта 's_name'	∅

3.16 Алгоритм конструктора класса cl_custom_5

Функционал: конструктор.

Параметры: p_main - указатель на главный объект, s_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 17.

Таблица 17 – Алгоритм конструктора класса cl_custom_5

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_5', который вызывает конструктор базового класса 'cl_custom_base' с передачей ему указателя на главный объект 'p_main' и имя объекта 's_name'	∅

3.17 Алгоритм конструктора класса cl_custom_6

Функционал: конструктор.

Параметры: p_main - указатель на главный объект, s_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 18.

Таблица 18 – Алгоритм конструктора класса cl_custom_6

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_6', который вызывает конструктор базового класса 'cl_custom_base' с передачей ему указателя на главный объект 'p_main' и имя объекта 's_name'	∅

3.18 Алгоритм метода build_tree_objects класса cl_custom_application

Функционал: создание иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода build_tree_objects класса cl_custom_application

№	Предикат	Действия	№ перехода
1		инициализация целочисленной переменной i = 0	2
2		вывод на экран "Object tree"	3
3		инициализация строковых переменных path_custom, sub_name	4
4		инициализация целочисленной переменной class_number	5
5		ввод значения переменной sub_name	6
6		вызов метода set_custom_name для текущего объекта this, передавая ему sub_name в качестве аргумента	7
7		объявление указателя parent на объект класса cl_custom_base	8
8		создание указателя last_created, который указывает на текущий объект this, который является экземпляром класса cl_custom_application	9
9		ввод значения и сохранение его в переменную path_custom	10
10	path_custom строка равна строке 'endtree'	ввод значения переменной sub_name, class_number	11
			∅
11		вызов метода get_object_path для объекта last_created , передавая ему строку path_custom, чтобы найти объект указанному в path_custom, полученный объект присваивается переменной parent	12

№	Предикат	Действия	№ перехода
12	parent равна nullptr	вызов метода print_custom_tree() для текущего объекта	13
			15
13		вывод на экран "The head object " , значение переменной path_custom, вывод " is not found"	14
14		вызов функции exit(1)	15
15	объект с именем sub_name существует у parent	вывод на экран значения переменной path_custom, вывод "Dubbing the names of subordinate objects"	22
			16
16	значения переменной class_number = 1	создание нового объекта типа cl_custom_application с родителем parent и присвоение указателя на него переменной last_created	22
			17
17	значения переменной class_number = 2	создание нового объекта типа cl_custom_2 с родителем parent и именем sub_name, затем присвоение указателя на него переменной last_created	22
			18
18	значения переменной class_number = 3	создание нового объекта типа cl_custom_ 3 с родителем parent и именем sub_name, затем присвоение указателя на него переменной last_created	22
			19
19	значения переменной class_number = 4	создание нового объекта типа cl_custom_4 с родителем parent и именем sub_name, затем присвоение указателя на него переменной	22

№	Предикат	Действия	№ перехода
		last_created	
			20
20	значения переменной class_number = 5	создание нового объекта типа cl_custom_5 с родителем parent и именем sub_name, затем присвоение указателя на него переменной last_created	22
			21
21	значения переменной class_number = 6	создание нового объекта типа cl_custom_6 с родителем parent и именем sub_name, затем присвоение указателя на него переменной last_created	22
			22
22		ввод значения переменной path_custom	∅

3.19 Алгоритм метода exec_custom_app класса cl_custom_application

Функционал: сборка приложения.

Параметры: отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения работы.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода exec_custom_app класса cl_custom_application

№	Предикат	Действия	№ перехода
1		инициализация строковых переменных command, input	2
2		создание указателя current_obj, который указывает на текущий объект, инициализируется значением this	3

№	Предикат	Действия	№ перехода
3		создание указателя extra_obj, который указывает на текущий объект, инициализируется значением this	4
4		создание стека строк st	5
5		вызов метода print_custom_tree для вывод иерархического дерева объектов	6
6		ввод значения переменной command	7
7	переменная command равна строке END	ввод значения переменной input	8
			30
8	значение переменной command равно "SET"	поиск объектов в дереве по указанному пути input и присвоение его переменной extra_obj	9
			12
9	найден объект extra_obj в дереве по указанному пути	присвоение значение текущего объекта найденному объекту	10
			11
10		вывод на экран "Object is set: ", вызов метода get_custom_name для текущего объекта и возврат его имени	12
11	не найден объект extra_obj в дерево по указанному пути	вывод на экран "The object was not found at the specified coordinate: " , вывод значения input	12
			12
12	значение переменой равно "FIND"	вызов метода get_object_path для текущего объекта с аргументом input, который представляет путь к другому объекту в структуре, полученный объект присваивается переменной extra_obj	13
			15
13	найден объект extra_obj в дереве по указанному пути	вывод на экран значение переменной input, вывод " Object name: ", обращение к методу	15

№	Предикат	Действия	№ перехода
		get_custom_name объект extra_obj	
			14
14	не найден объект extra_obj в дерево по указанному пути	вывод на экран значение переменной input, вывод " Object is not found"	15
			15
15	значение переменой равно "MOVE"	вызов метода get_object_path для текущего объекта с аргументом input, который представляет путь к другому объекту в структуре, полученный объект присваивается переменной extra_obj	16
			20
16	установлен новый главный объект с помощью вызова метода set_head_object	вывод на экран "New head object: ", обращение к методу get_custom_name объект extra_obj	20
			17
17	не найден объект extra_obj в дерево по указанному пути	вывод на экран значения переменной input, вывод " Head object is not found"	20
			18
18	объект с именем текущего объекта в качестве подчиненного объекта у объекта extra_obj существует	вывод на экран значения переменной input, вывод Dubbing the names of subordinate objects"	20
			19
19	объект с именем текущего объекта в качестве подчиненного объекта у объекта extra_obj не существует	вывод на экран значения переменной input, вывод " Redefining the head object failed"	20
			20
20	значение переменой равно "DELETE"	присвоение переменной extra_obj указатель на подчиненный объект с именем, которое было	21

№	Предикат	Действия	№ перехода
		введено	
			29
21	найден объект extra_obj в дереве по указанному пути		22
			29
22	объект extra_obj не является корневым объектом	добавление имени объектов в стек st, начиная с объекта extra_obj, и двигаясь вверх по иерархии до корневого объекта	23
			24
23		обновление переменной extra_obj, чтобы она указывала на родительский объект текущего объекта extra_obj	24
24		вызов метода удаления подчиненного объекта с именем input, из текущего объекта current_obj	25
25		вывод на экран "The object "	26
26	стек st не пустой	вывод на экран символ '/' , вывод на экран верхний элемент стека st	27
			28
27		удаление этого элемента из стека	28
28		вывод на экран " has been deleted"	29
29		ввод значения переменной command	30
30		вывод на экран "Current object hierarchy tree	31
31		вызов метода print_custom_tree для текущего объекта	∅

3.20 Алгоритм конструктора класса cl_custom_application

Функционал: конструктор.

Параметры: p_main - указатель на объект класса 'cl_custom_base'.

Алгоритм конструктора представлен в таблице 21.

Таблица 21 – Алгоритм конструктора класса cl_custom_application

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_application', который вызывает конструктор базового класса 'cl_custom_base', передавая ему указатель p_main в качестве параметра	∅

3.21 Алгоритм функции main

Функционал: основной алгоритм работы программы.

Параметры: отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения работы программы.

Алгоритм функции представлен в таблице 22.

Таблица 22 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта 'obj_custom_app' класса 'cl_custom_application' с родительским объектом, указанным как 'nullptr'	2
2		вызов метода 'build_tree_objects()' для объекта 'obj_custom_app'	3
3		вызов метода 'exec_custom_app()' для объекта 'obj_custom_app' и возврат результата выполнения этого метода	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-26.

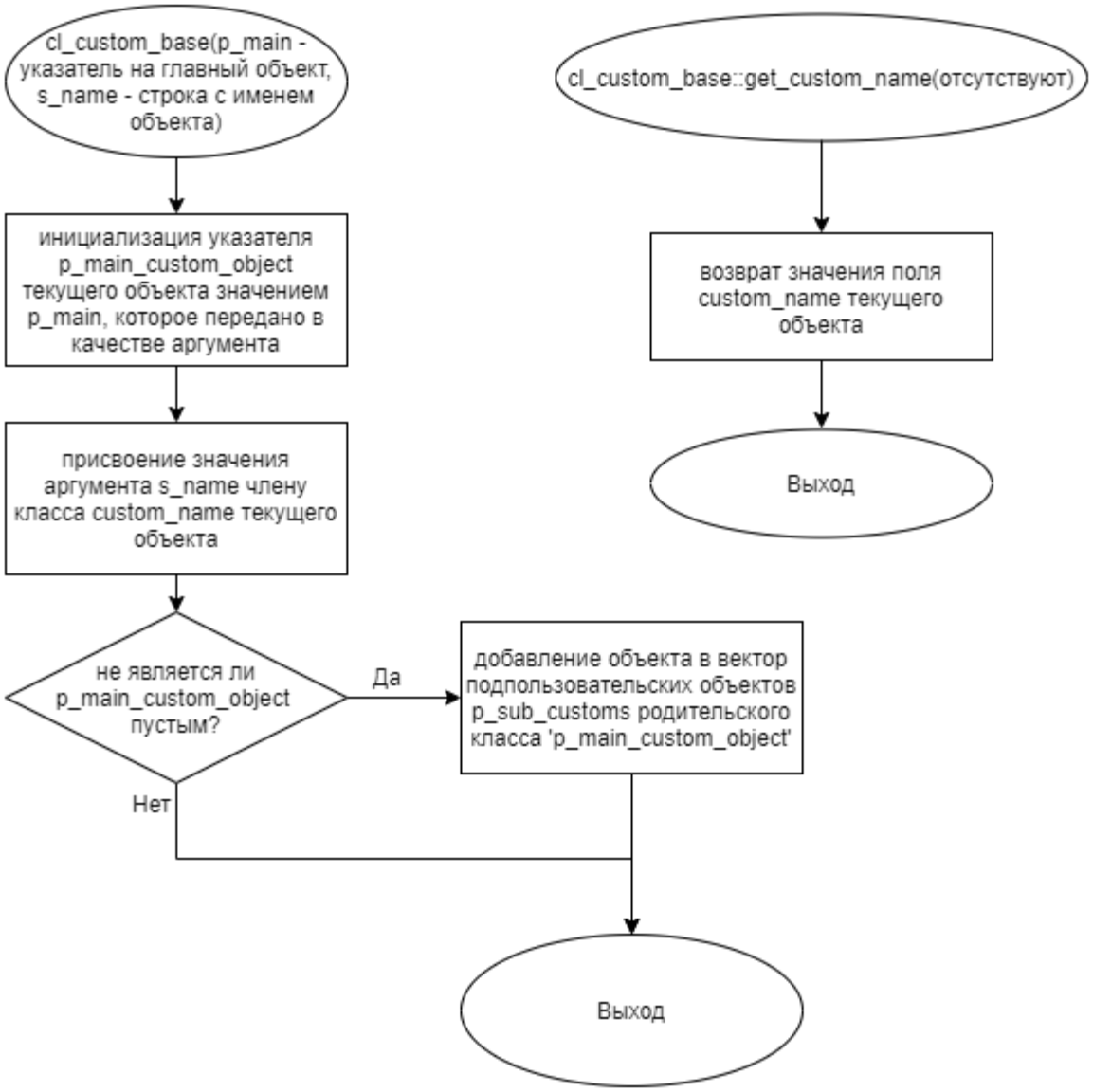

Рисунок 1 – Блок-схема алгоритма

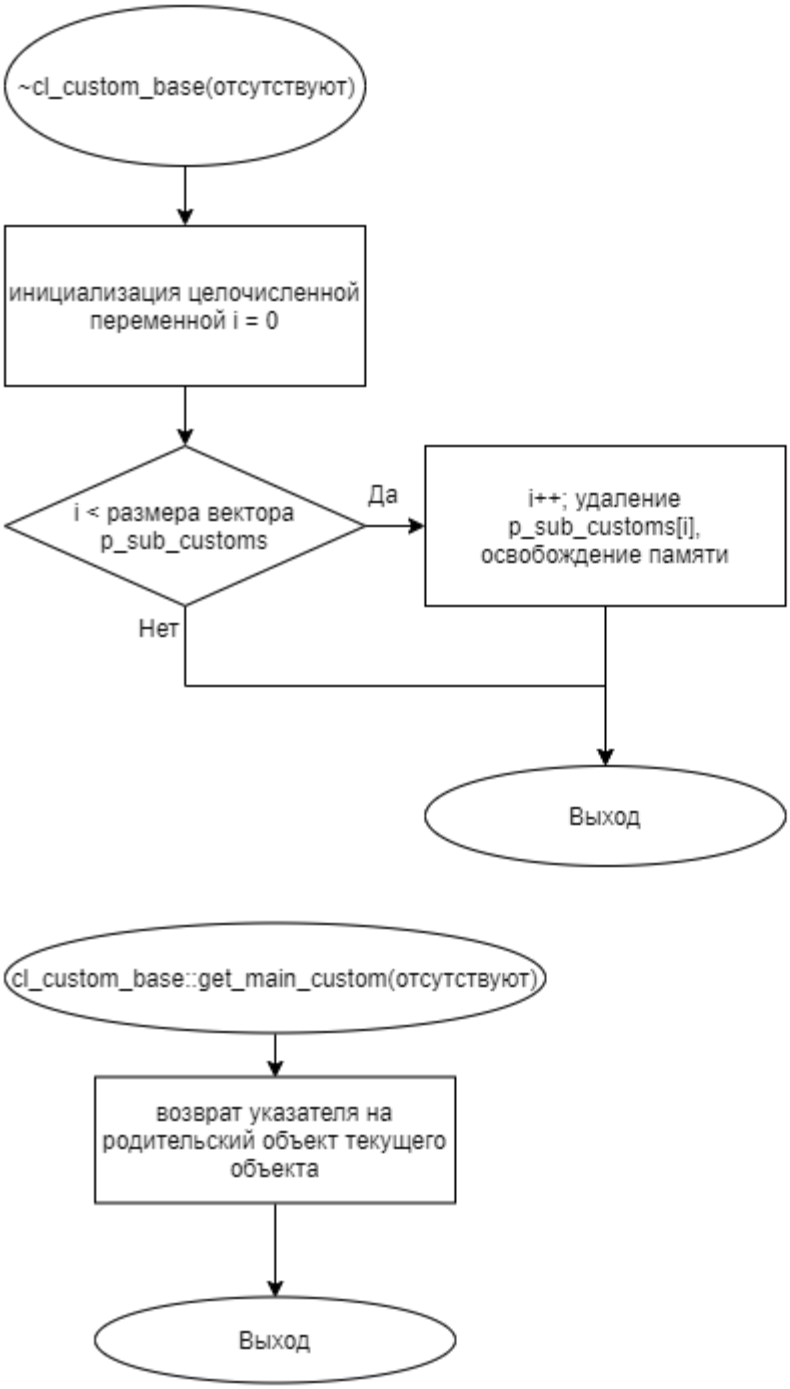

Рисунок 2 – Блок-схема алгоритма

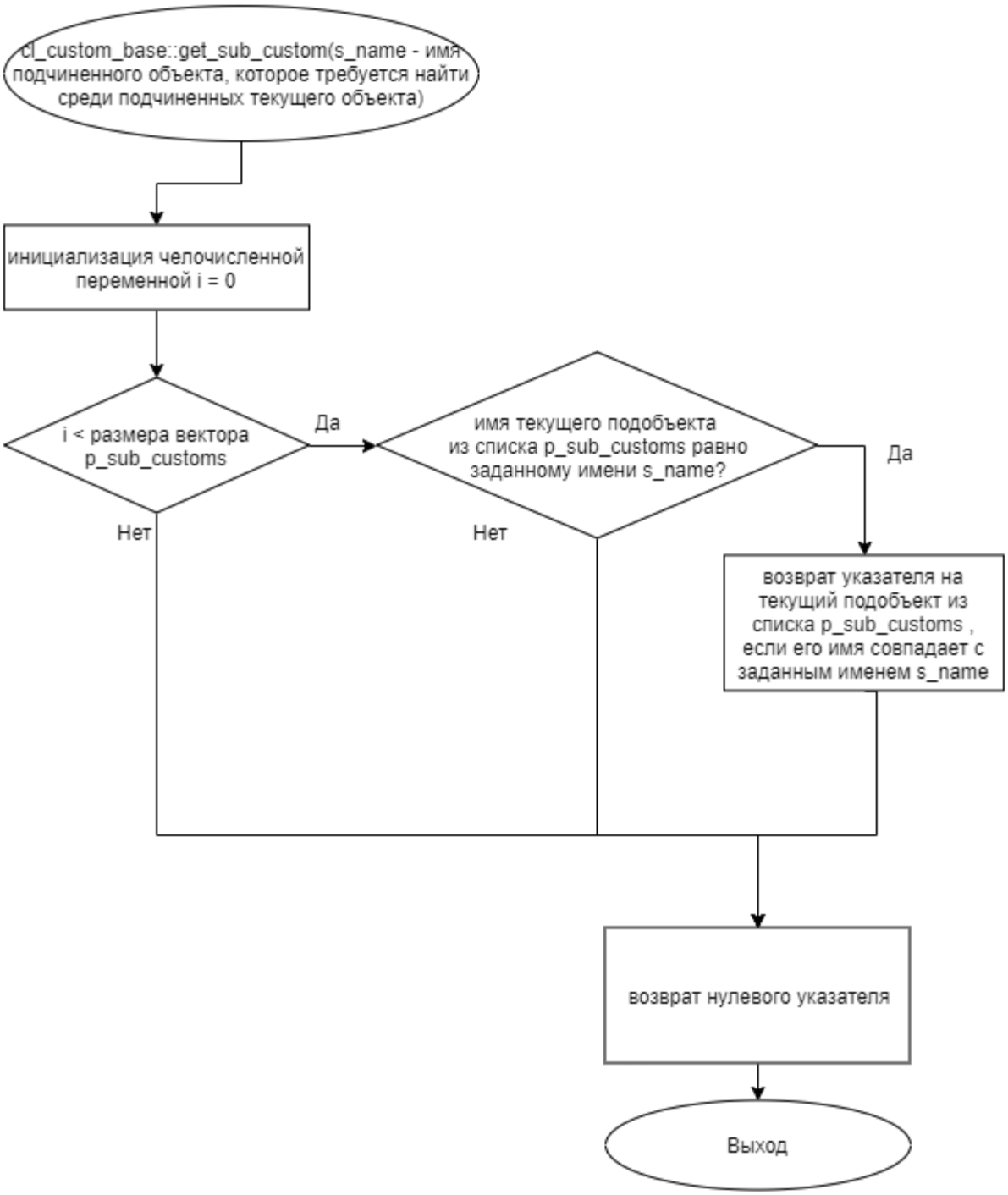

Рисунок 3 – Блок-схема алгоритма

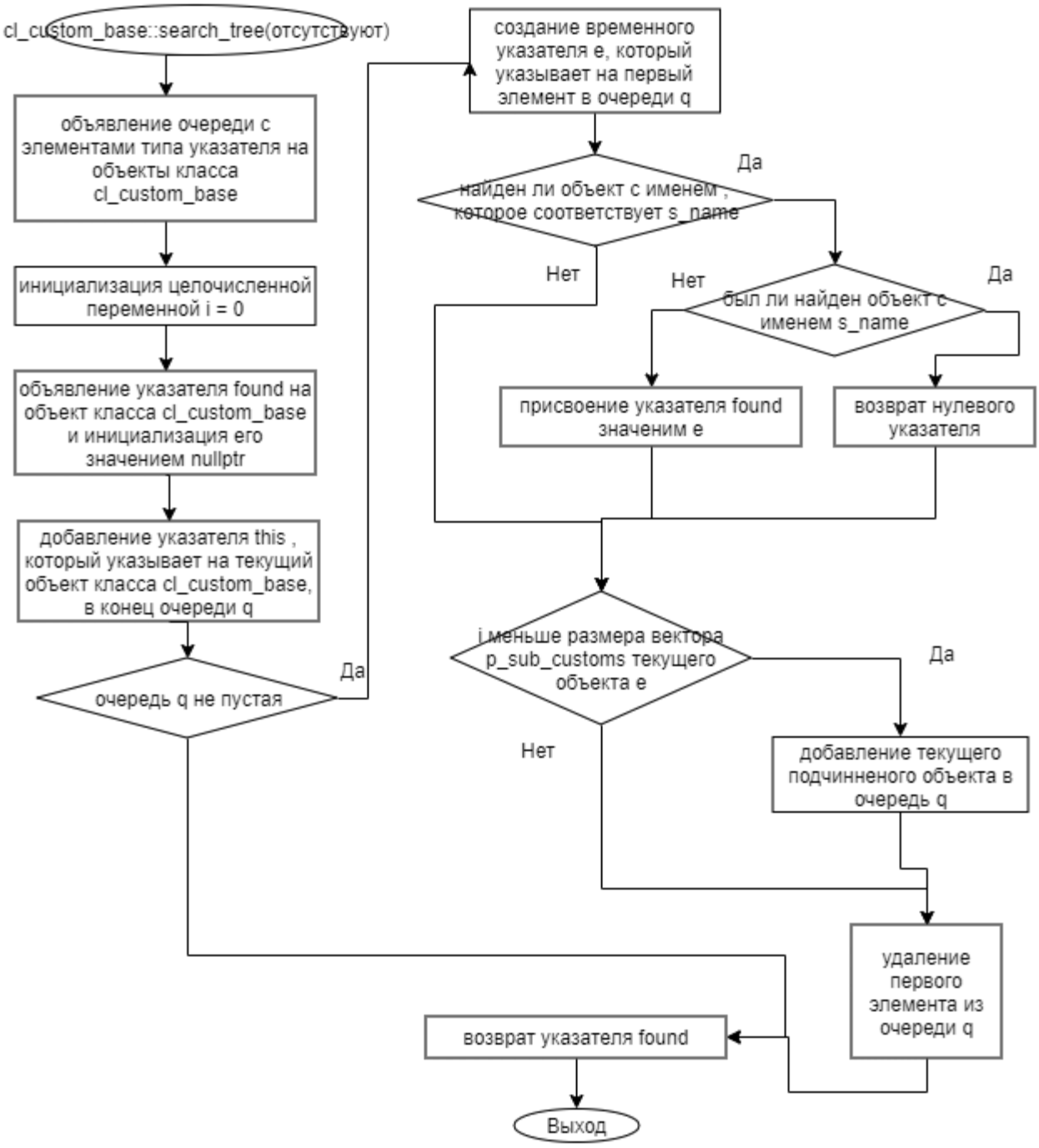

Рисунок 4 – Блок-схема алгоритма

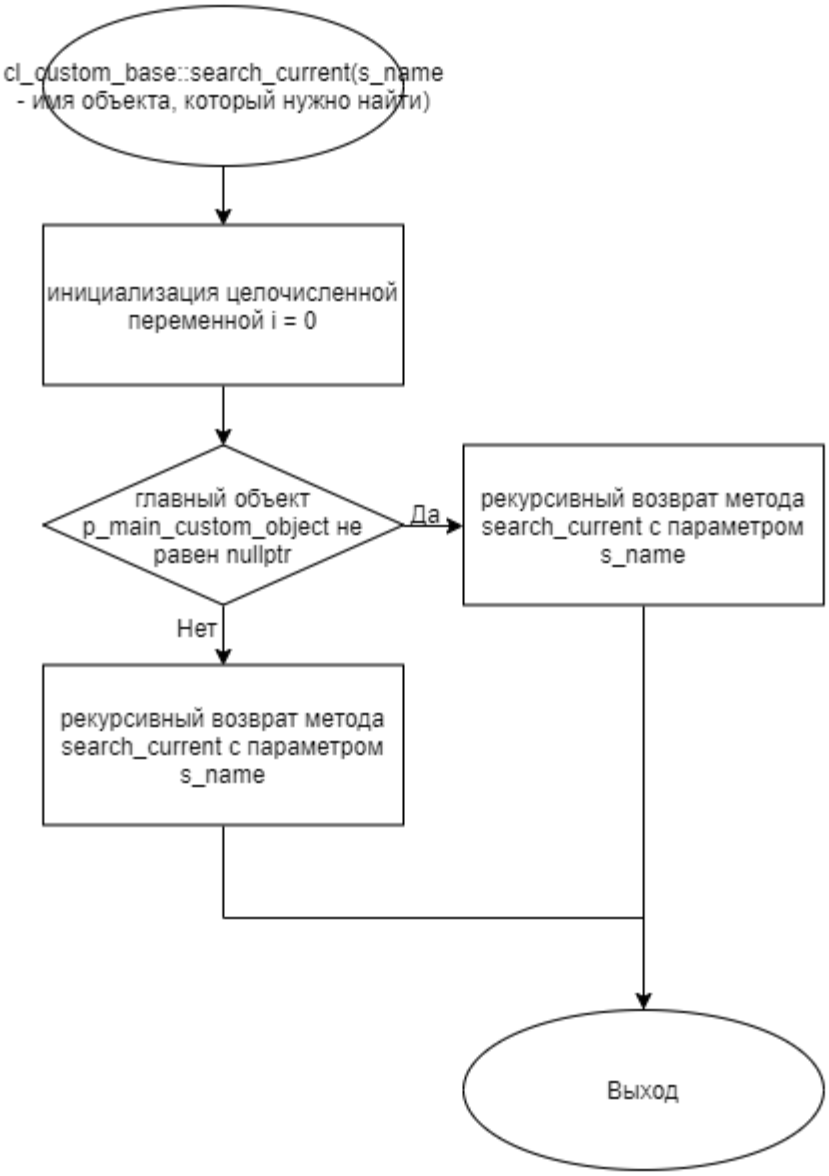

Рисунок 5 – Блок-схема алгоритма

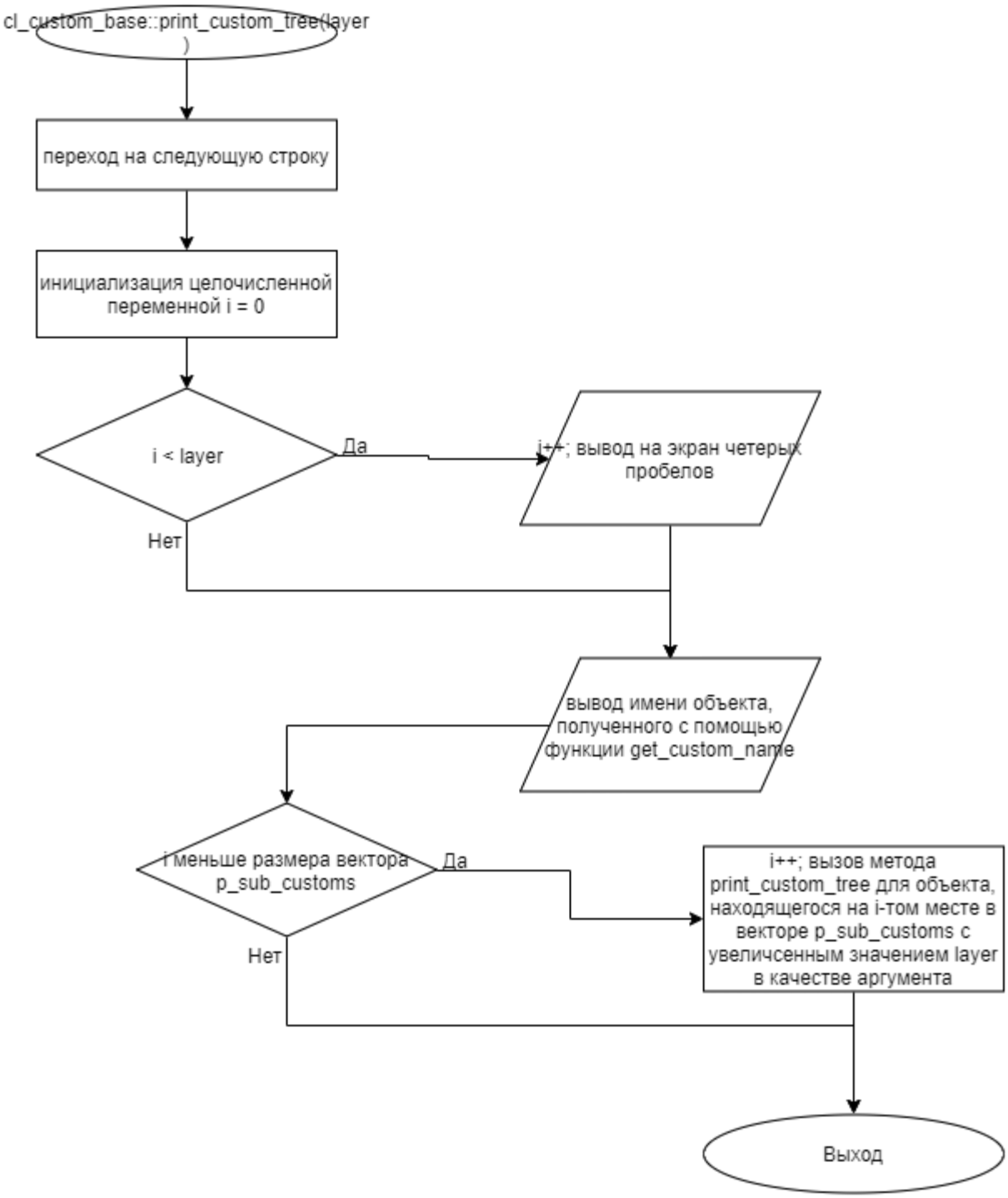

Рисунок 6 – Блок-схема алгоритма

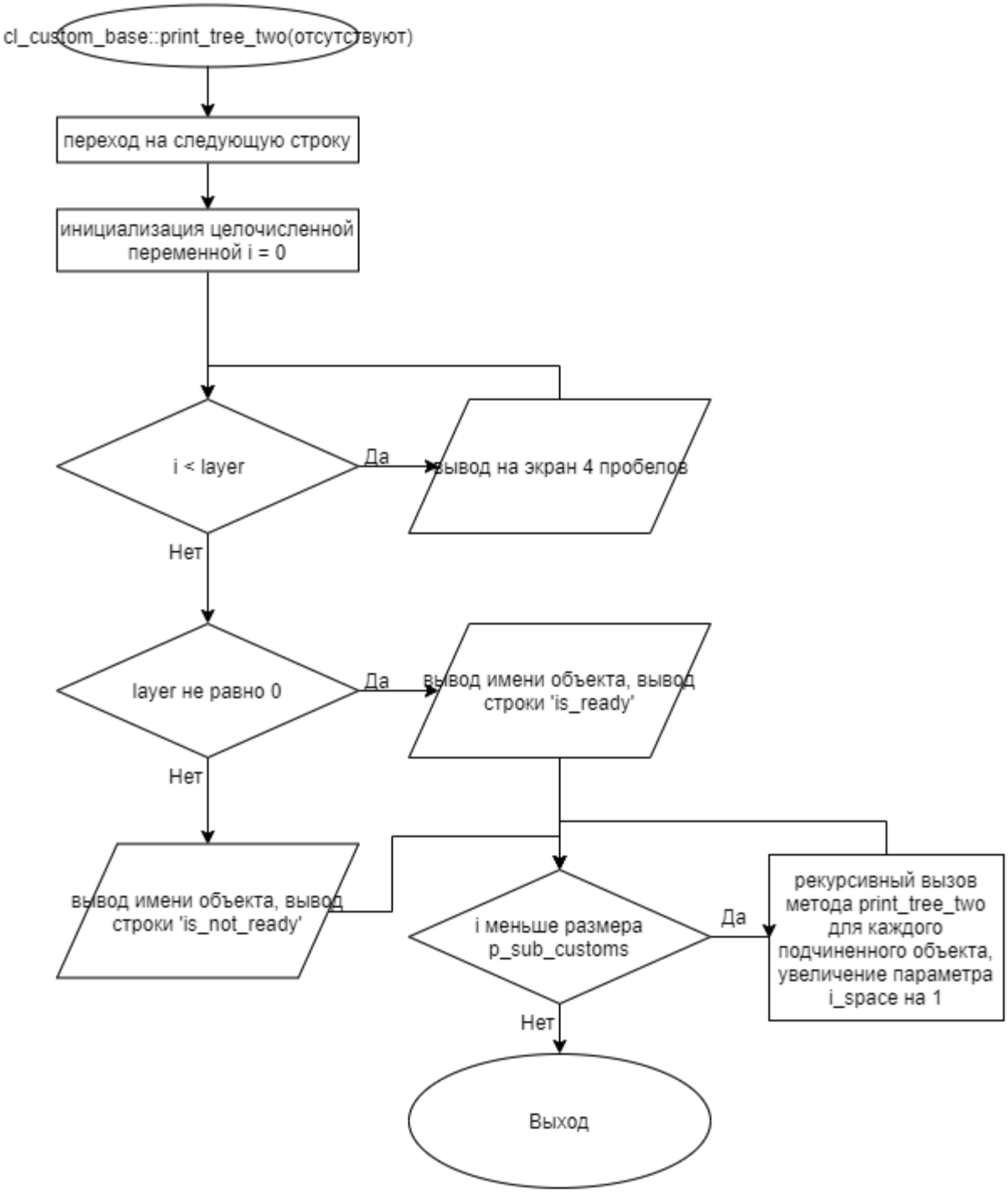

Рисунок 7 – Блок-схема алгоритма

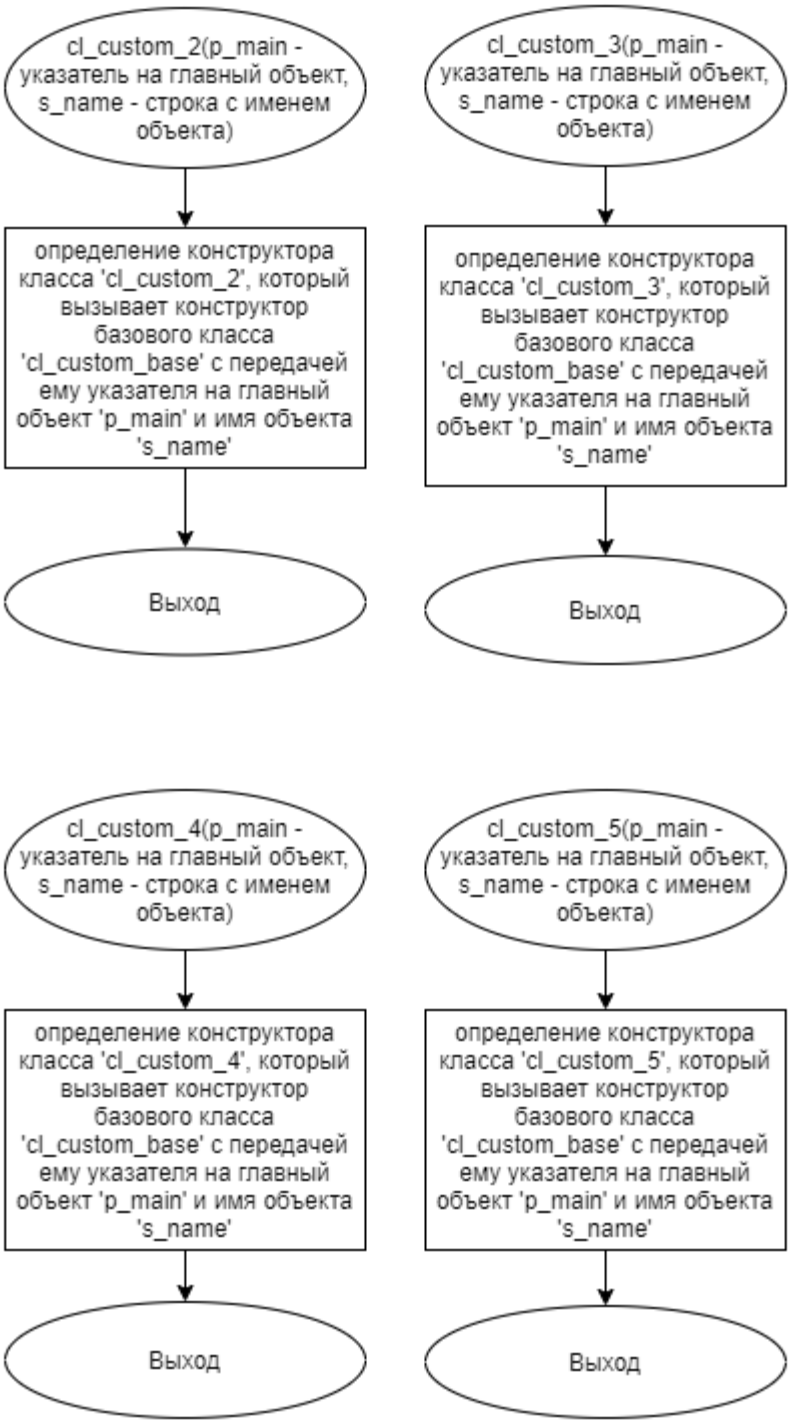

Рисунок 8 – Блок-схема алгоритма

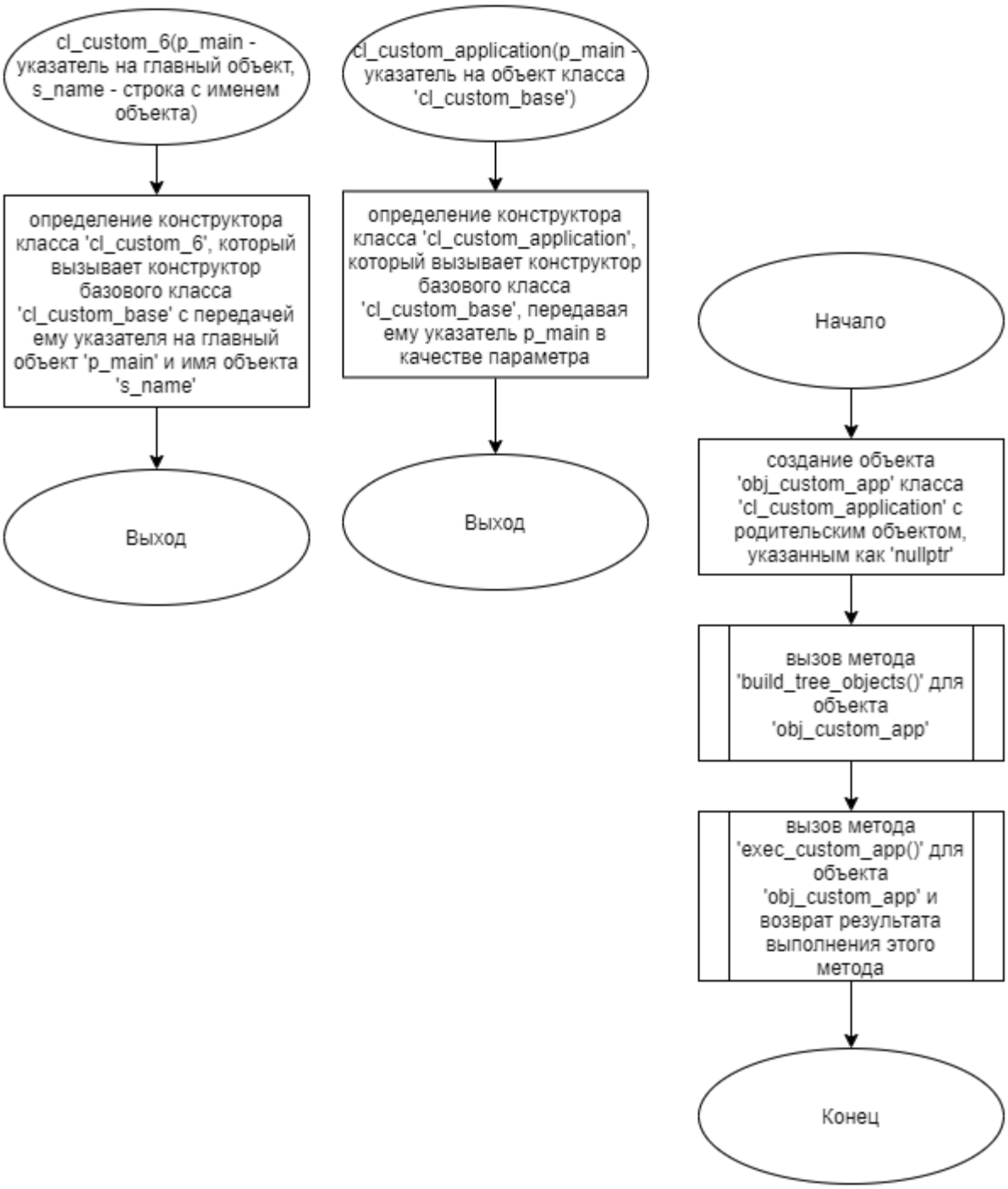

Рисунок 9 – Блок-схема алгоритма

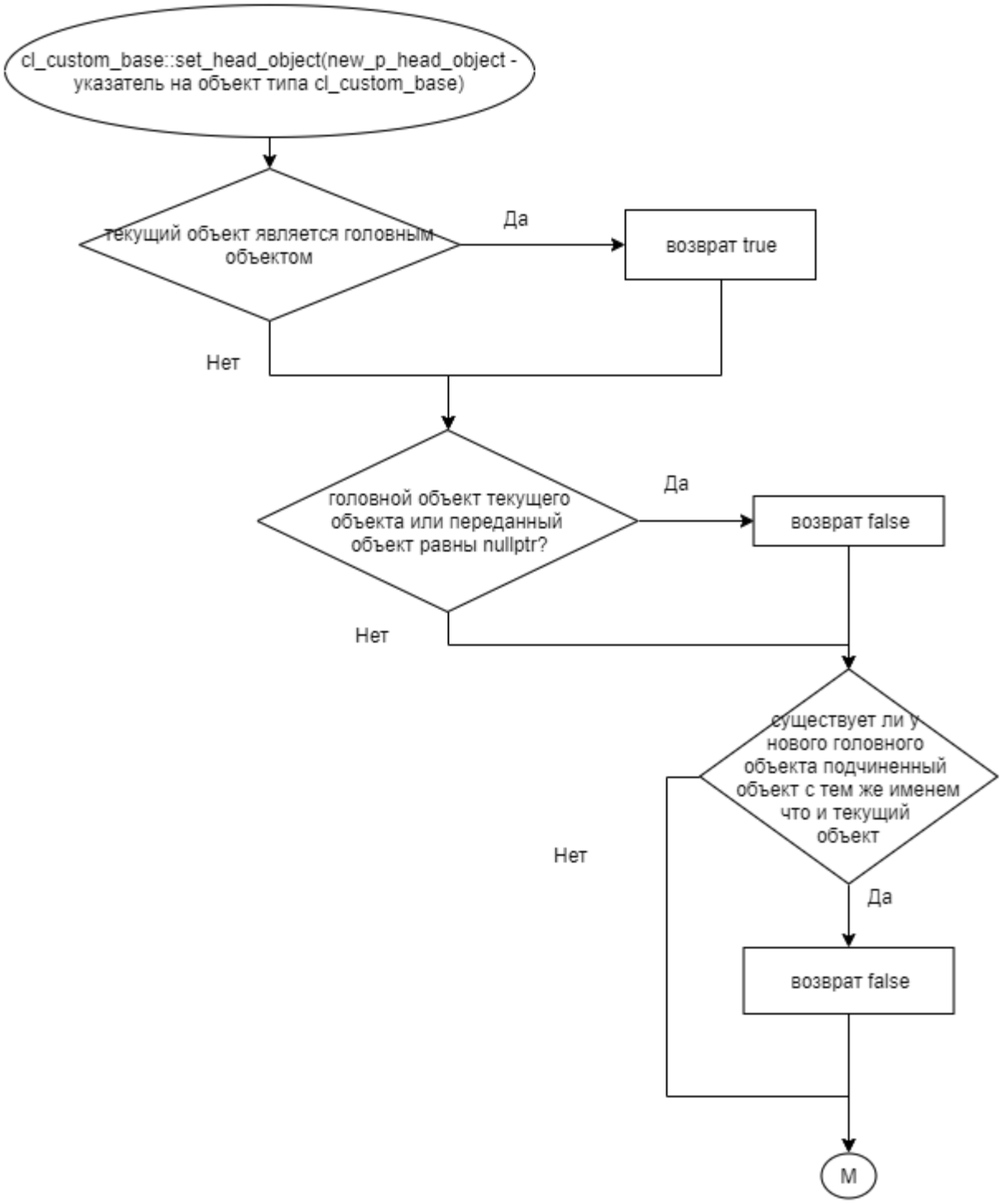

Рисунок 10 – Блок-схема алгоритма

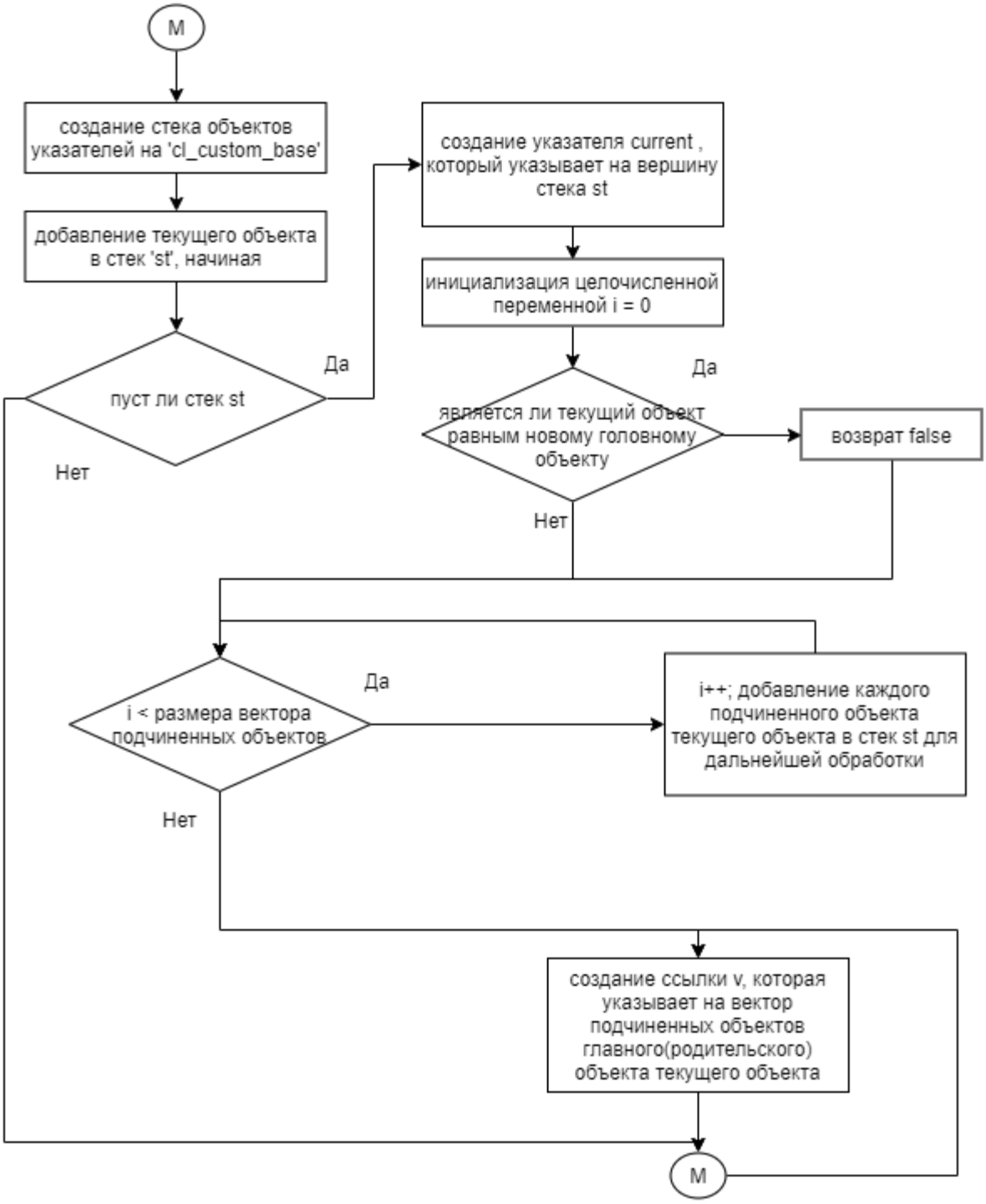

Рисунок 11 – Блок-схема алгоритма

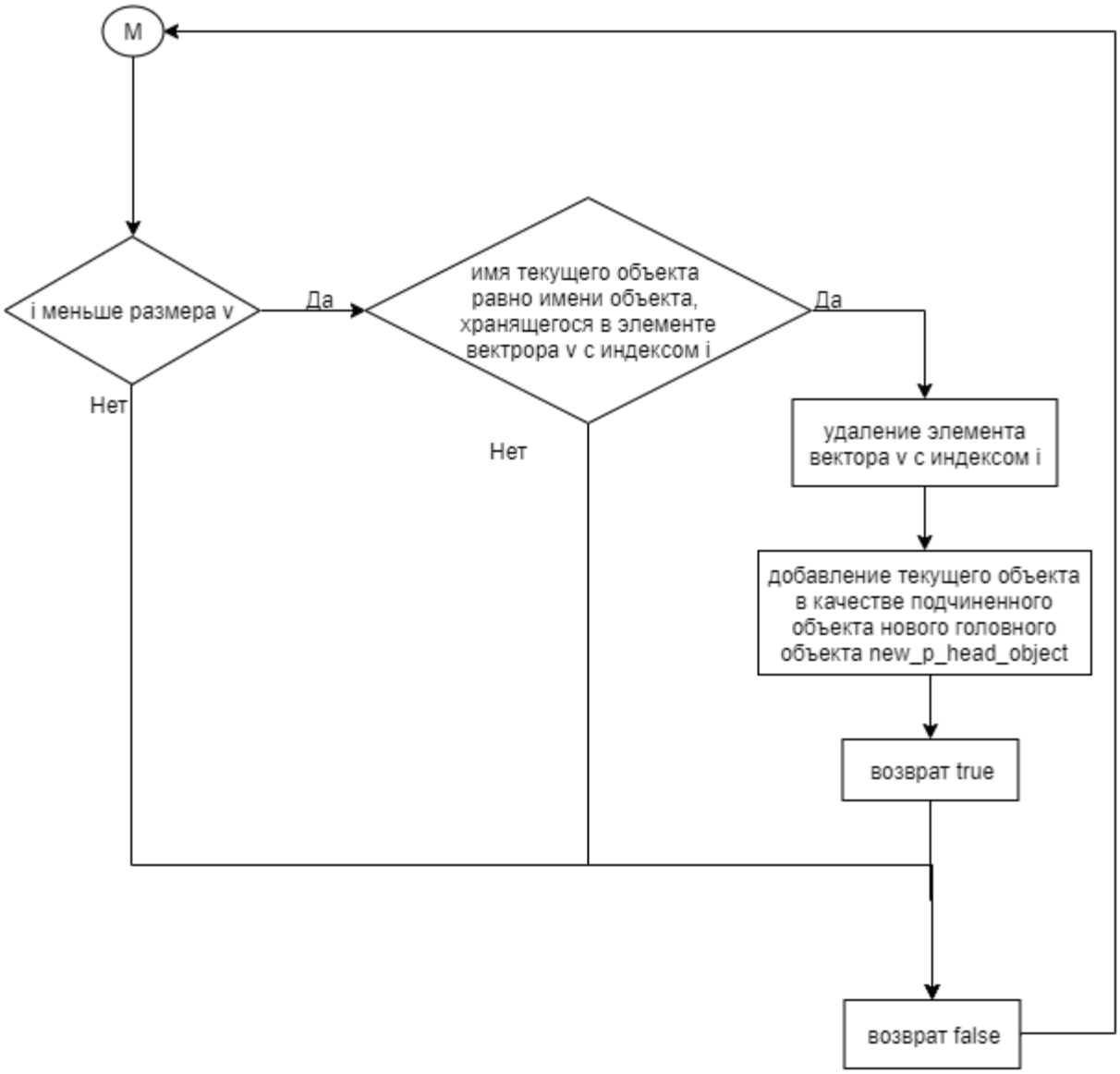

Рисунок 12 – Блок-схема алгоритма

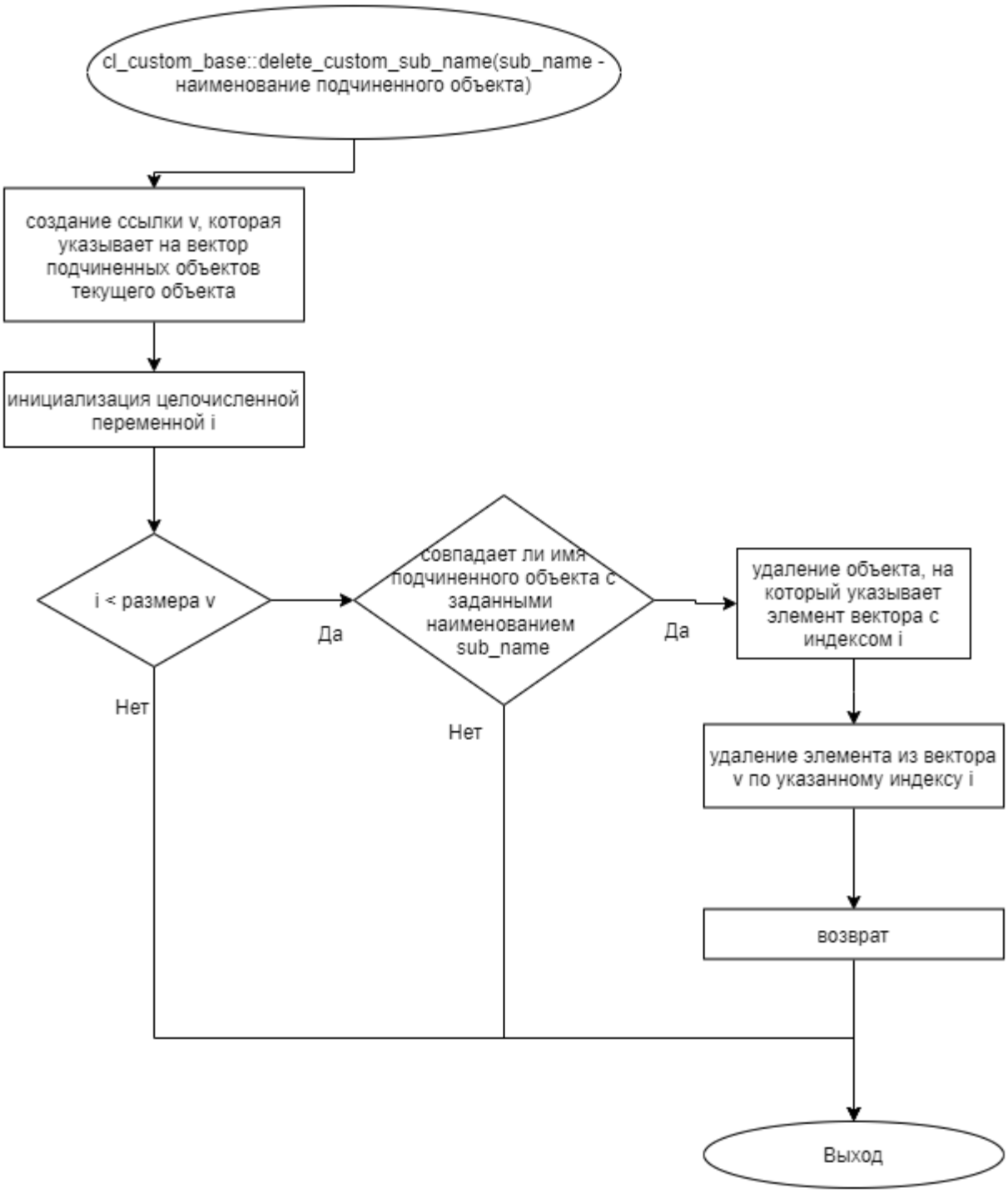

Рисунок 13 – Блок-схема алгоритма

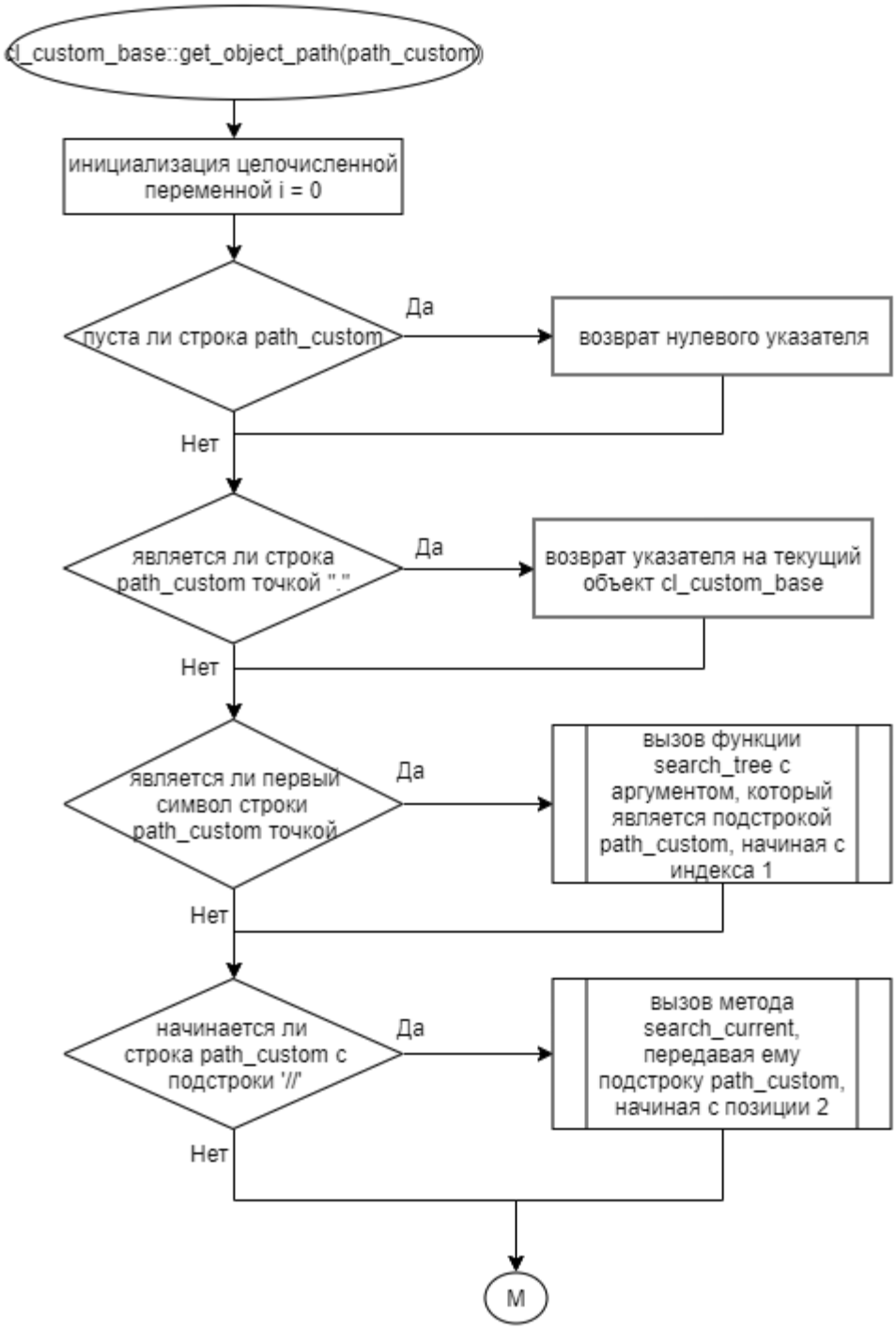

Рисунок 14 – Блок-схема алгоритма

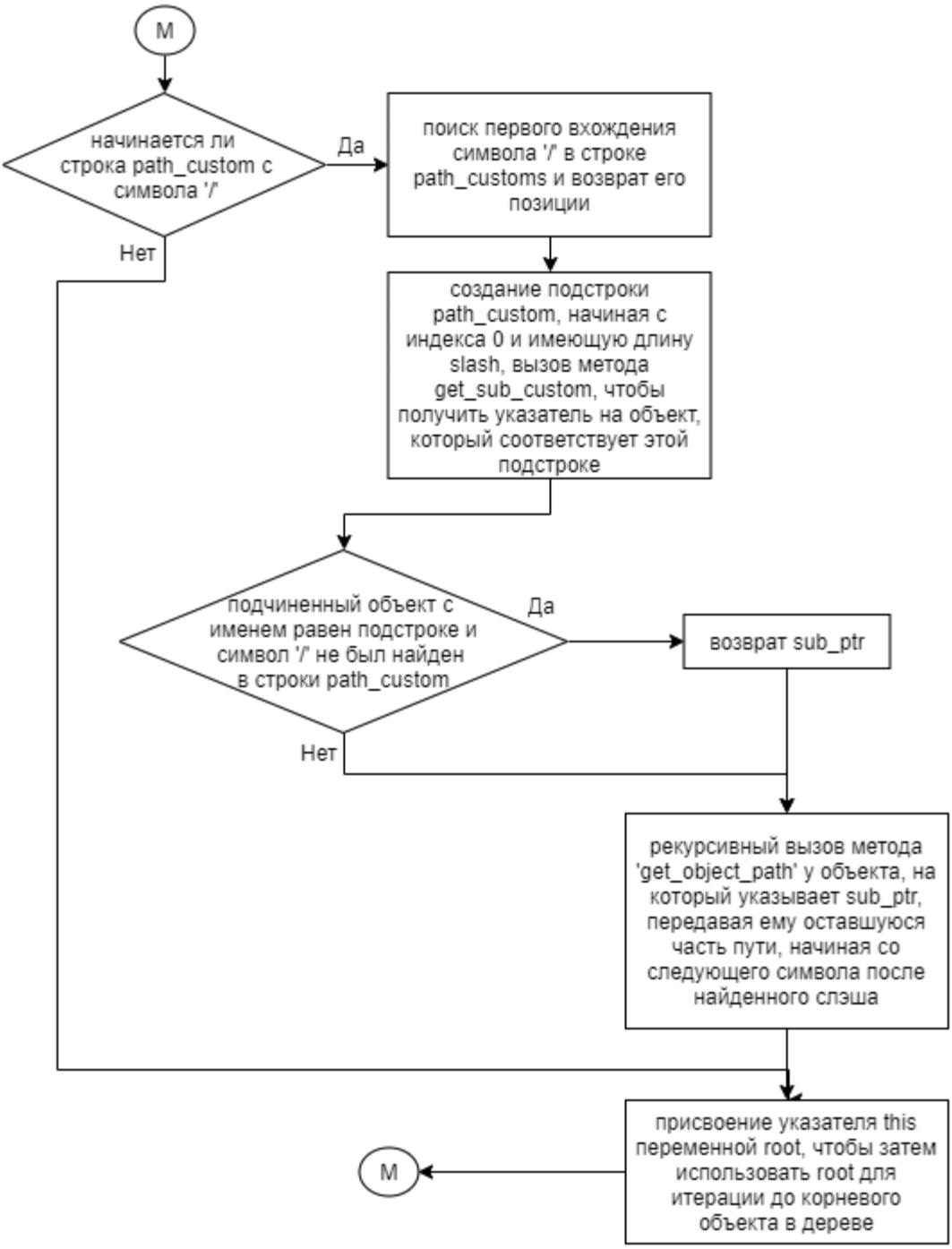

Рисунок 15 – Блок-схема алгоритма

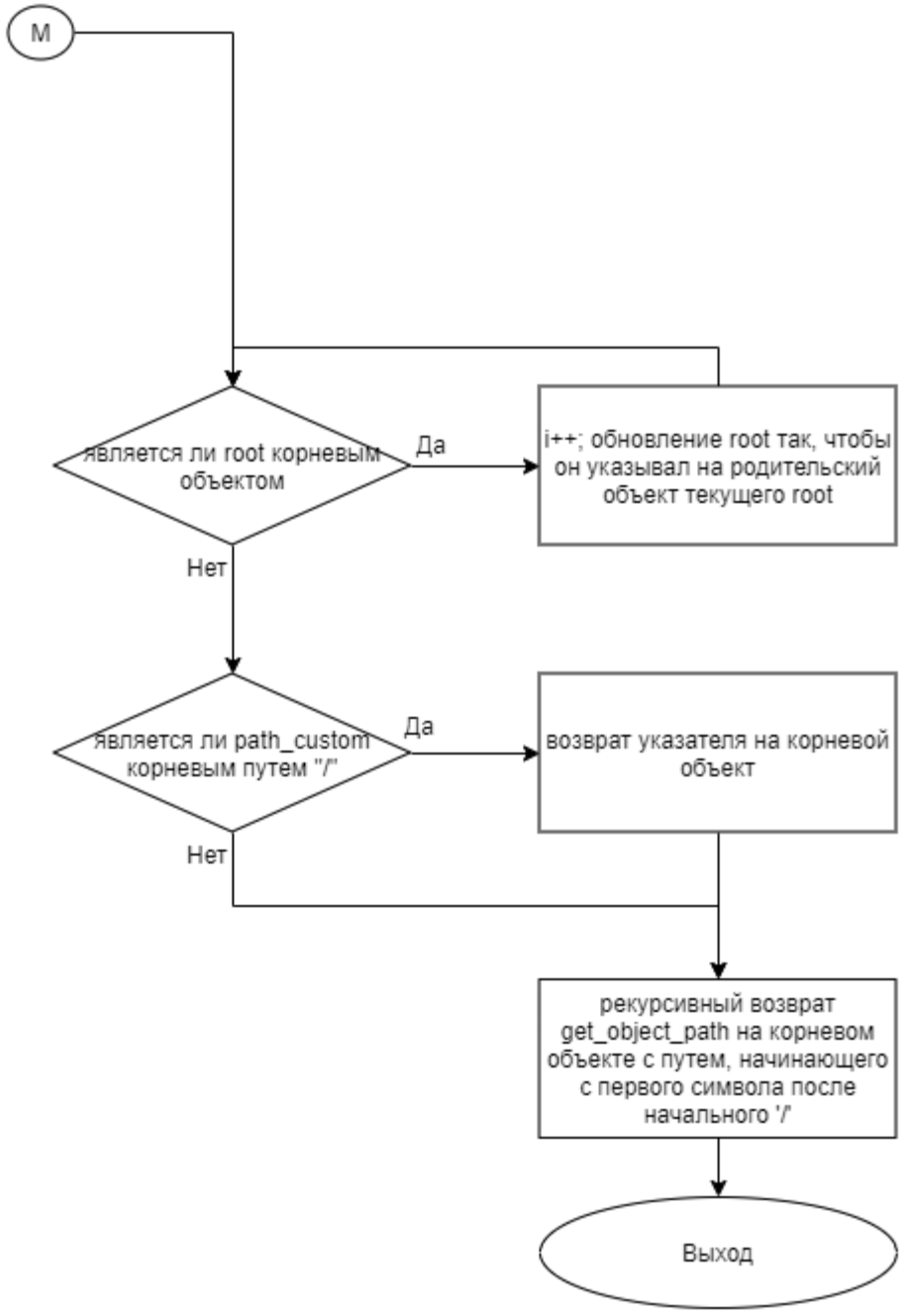

Рисунок 16 – Блок-схема алгоритма

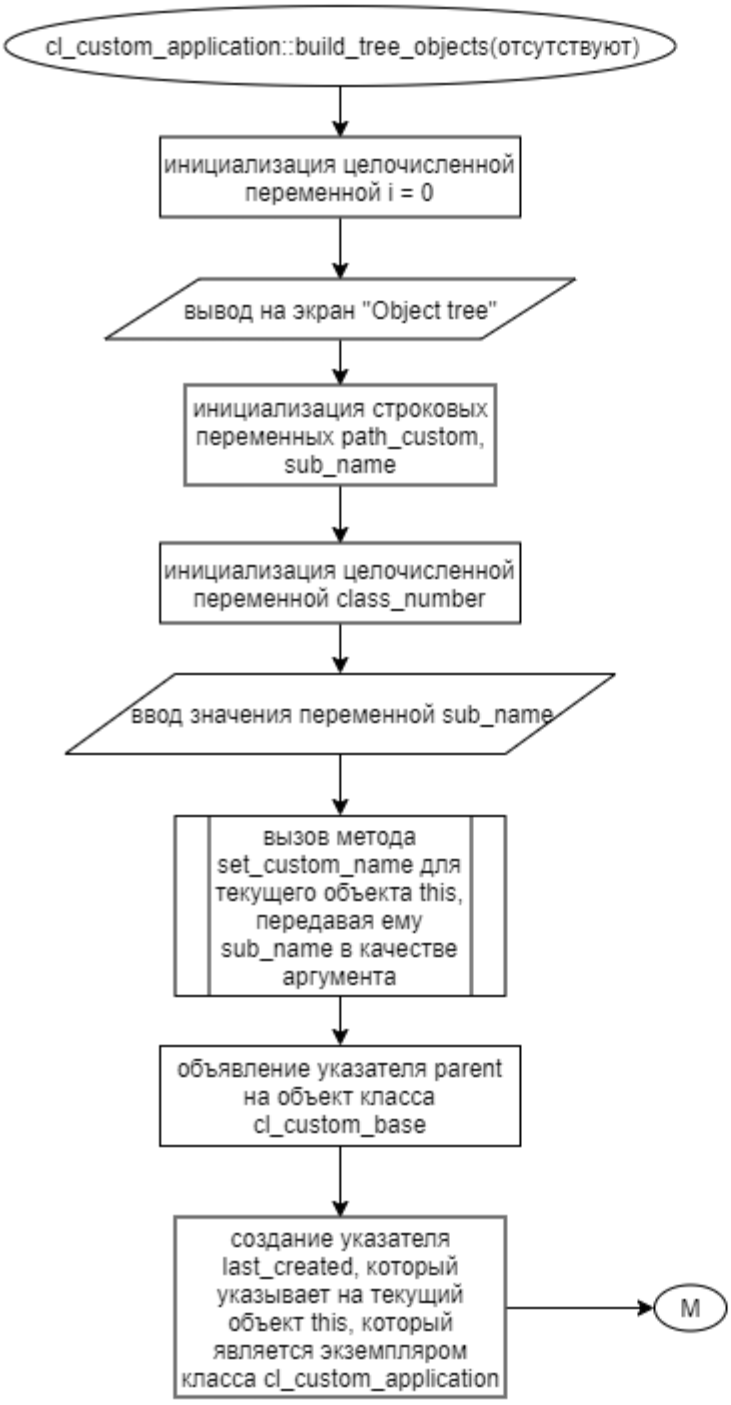

Рисунок 17 – Блок-схема алгоритма

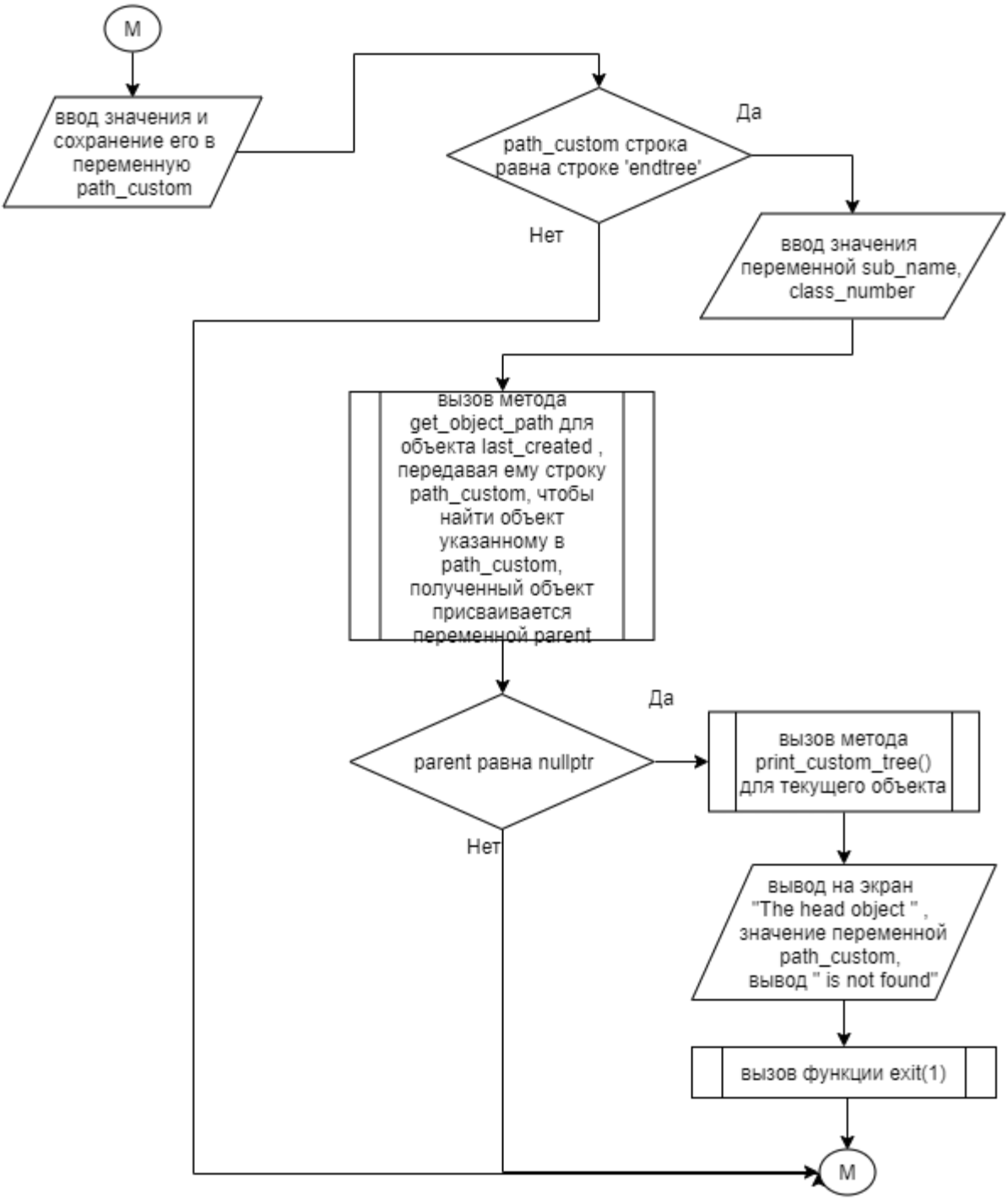

Рисунок 18 – Блок-схема алгоритма

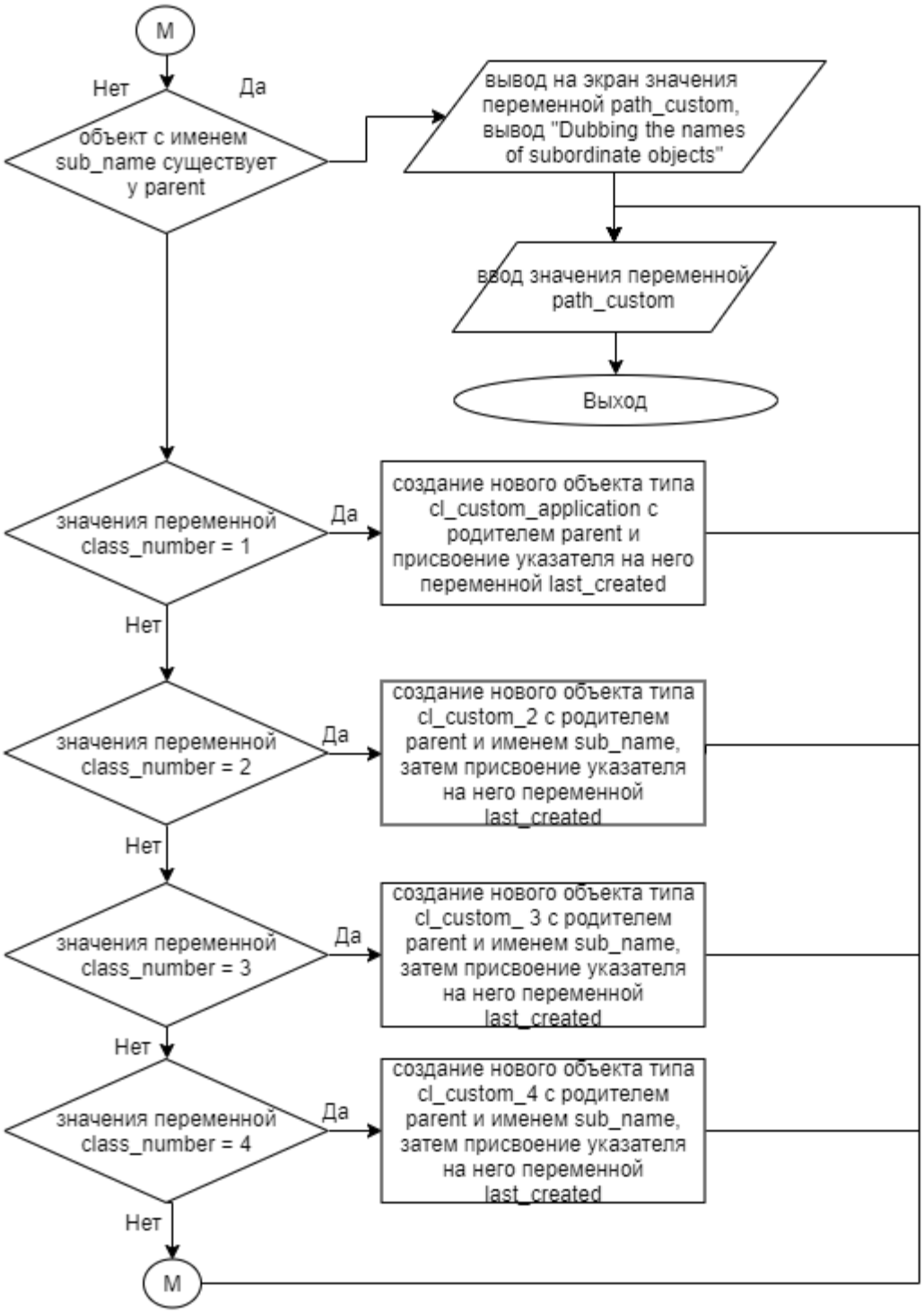

Рисунок 19 – Блок-схема алгоритма

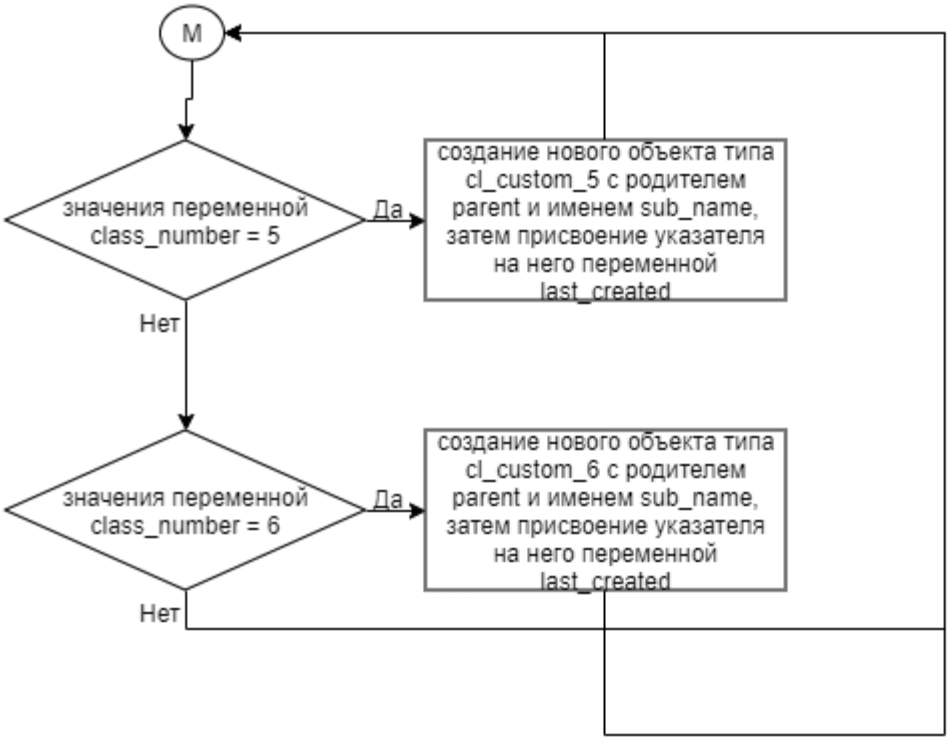

Рисунок 20 – Блок-схема алгоритма

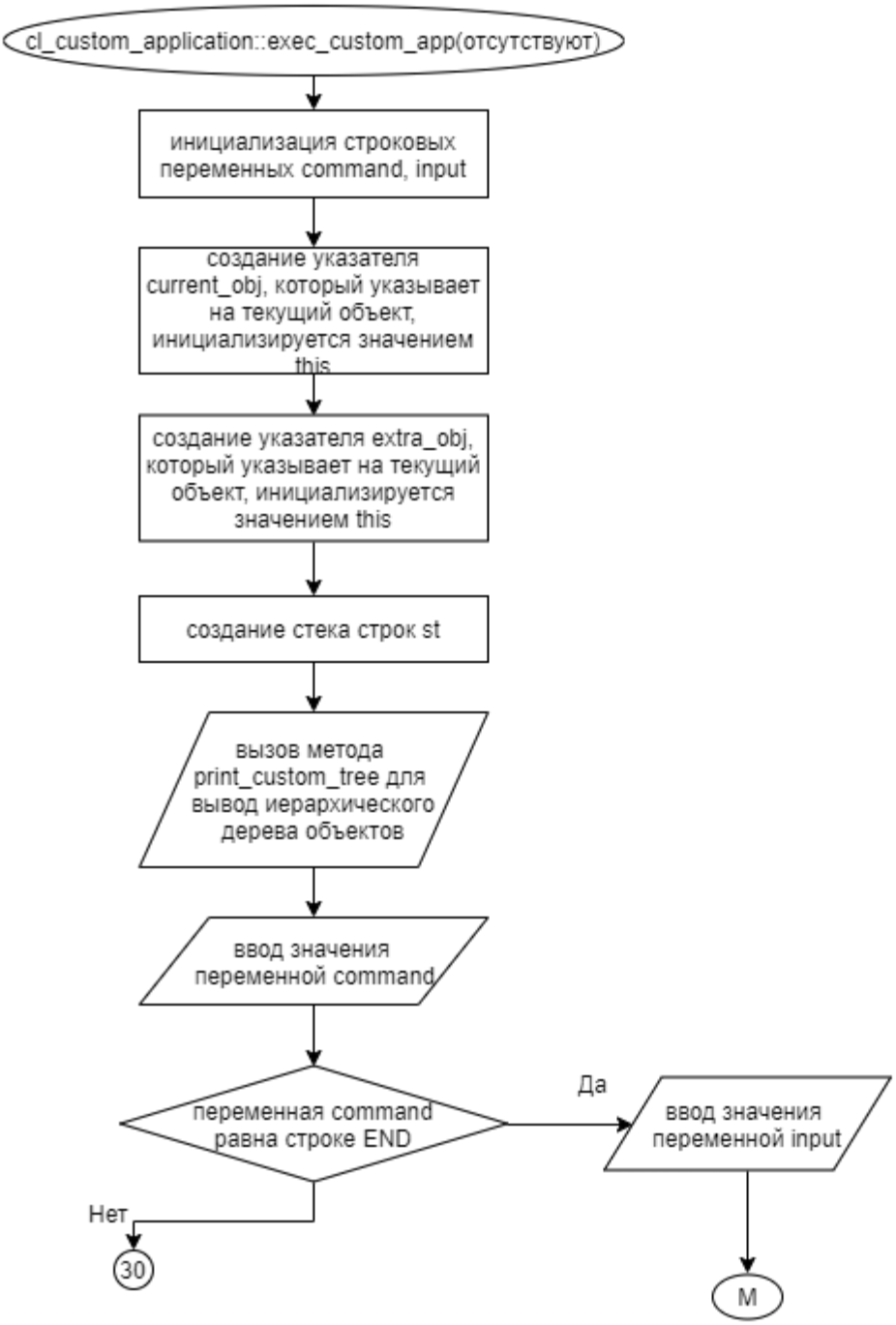

Рисунок 21 – Блок-схема алгоритма

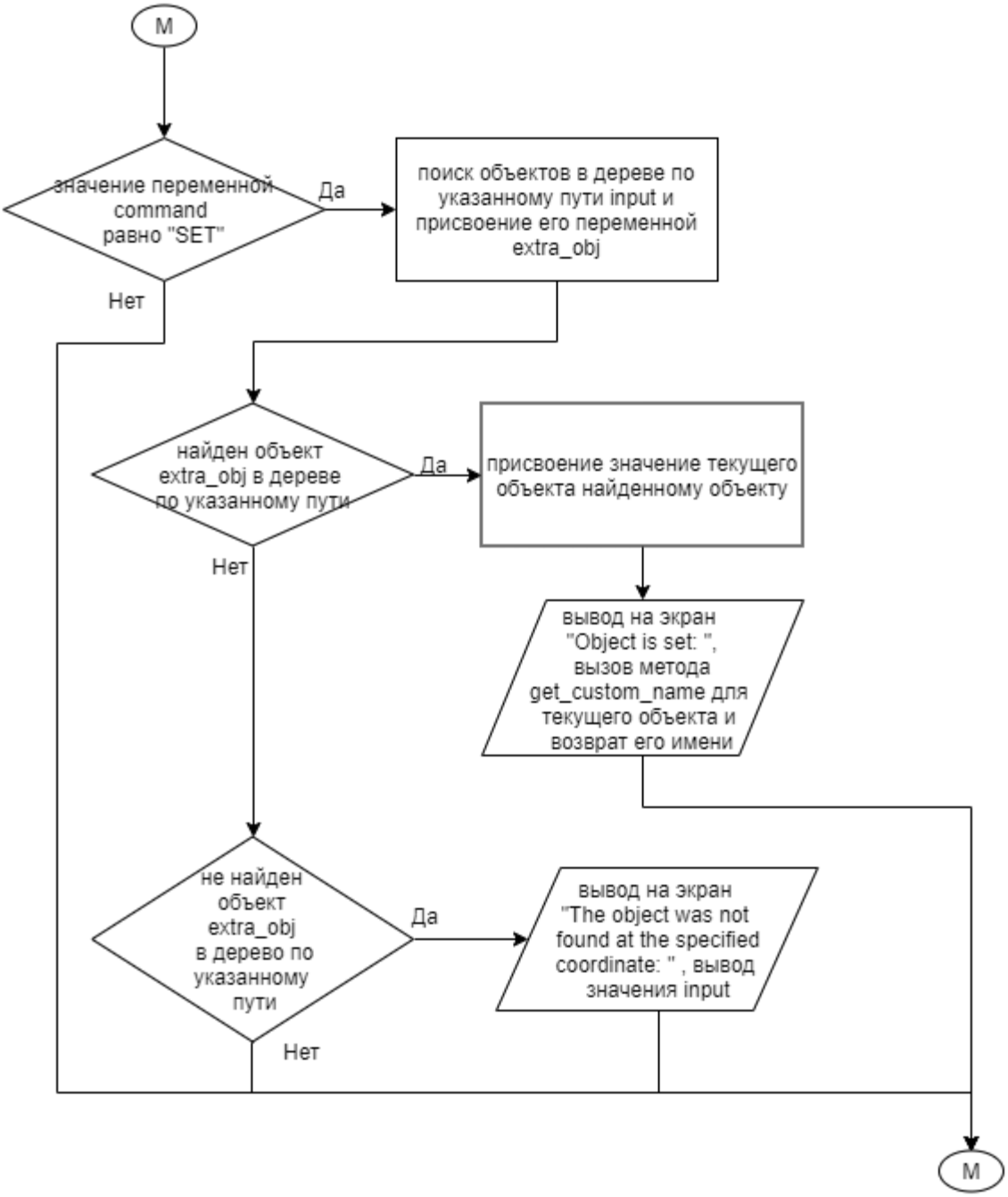

Рисунок 22 – Блок-схема алгоритма

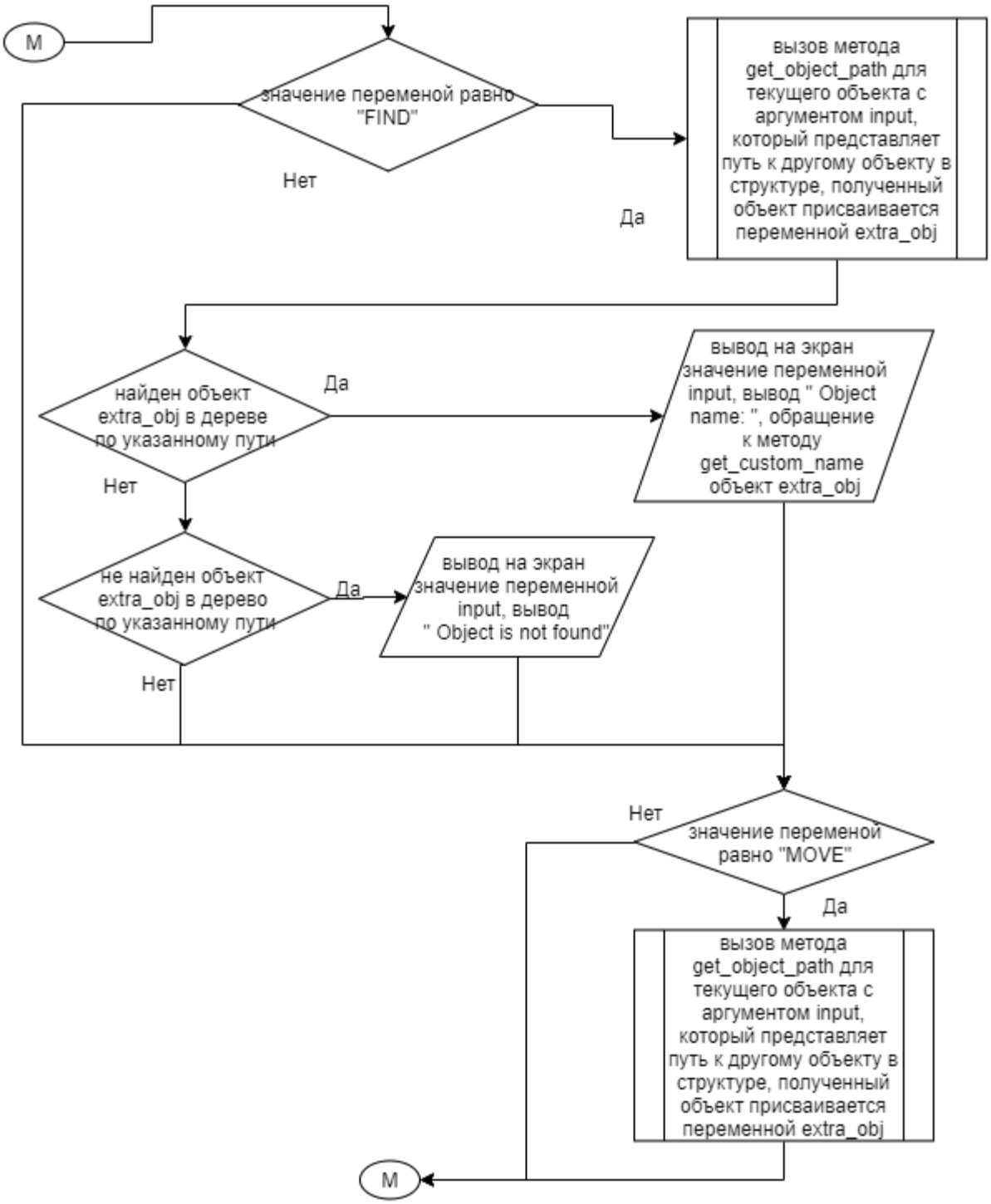

Рисунок 23 – Блок-схема алгоритма

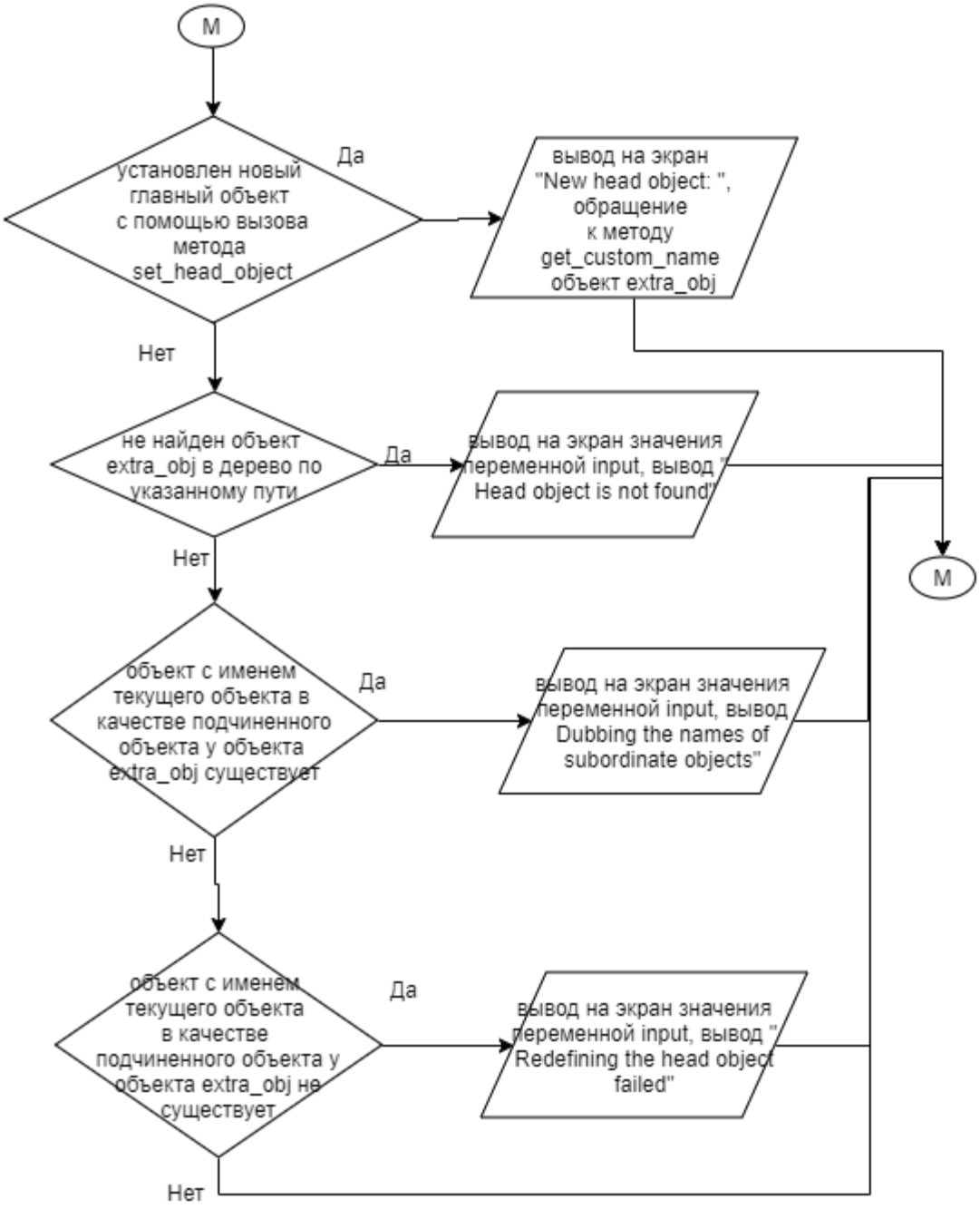

Рисунок 24 – Блок-схема алгоритма

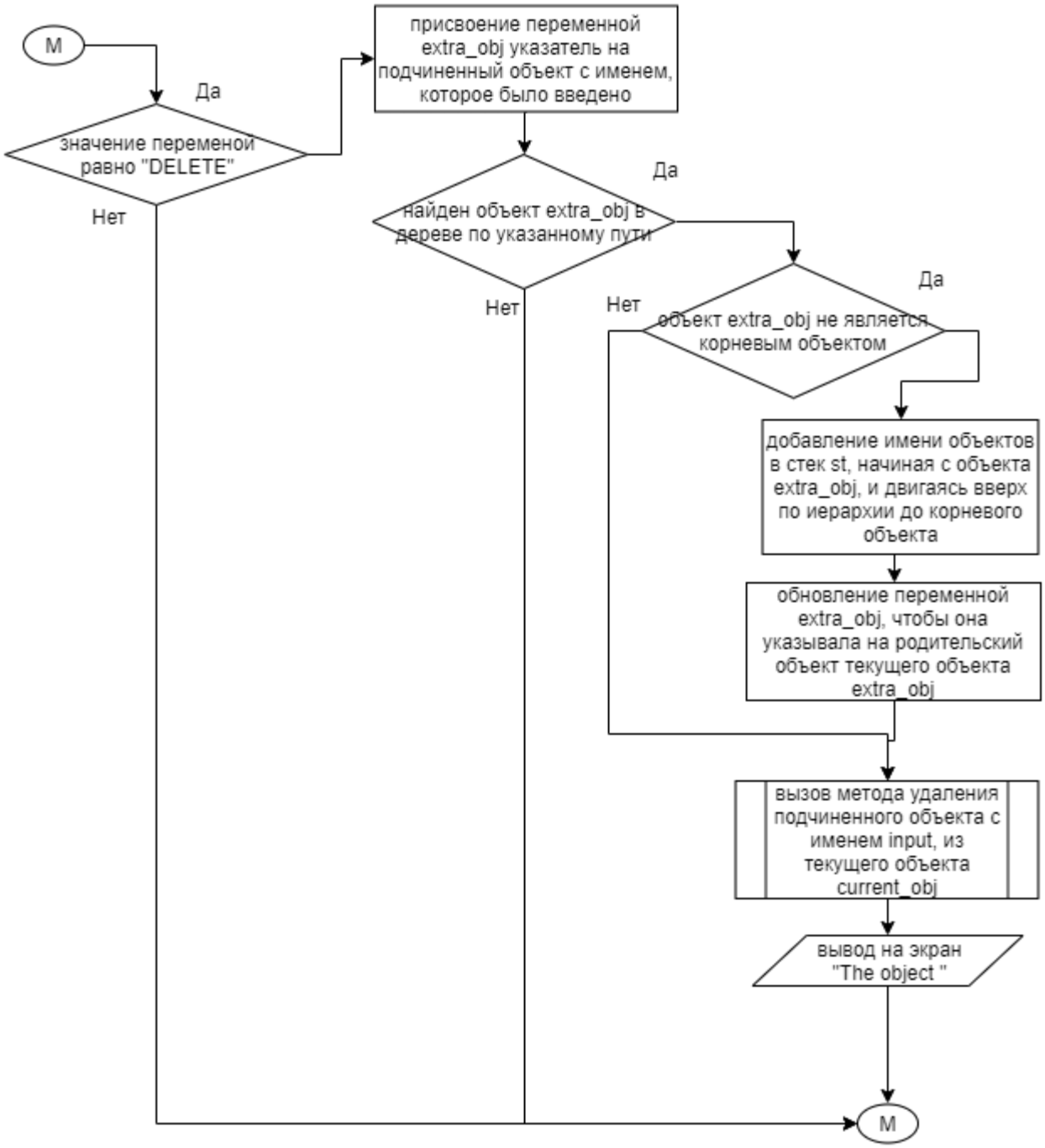

Рисунок 25 – Блок-схема алгоритма

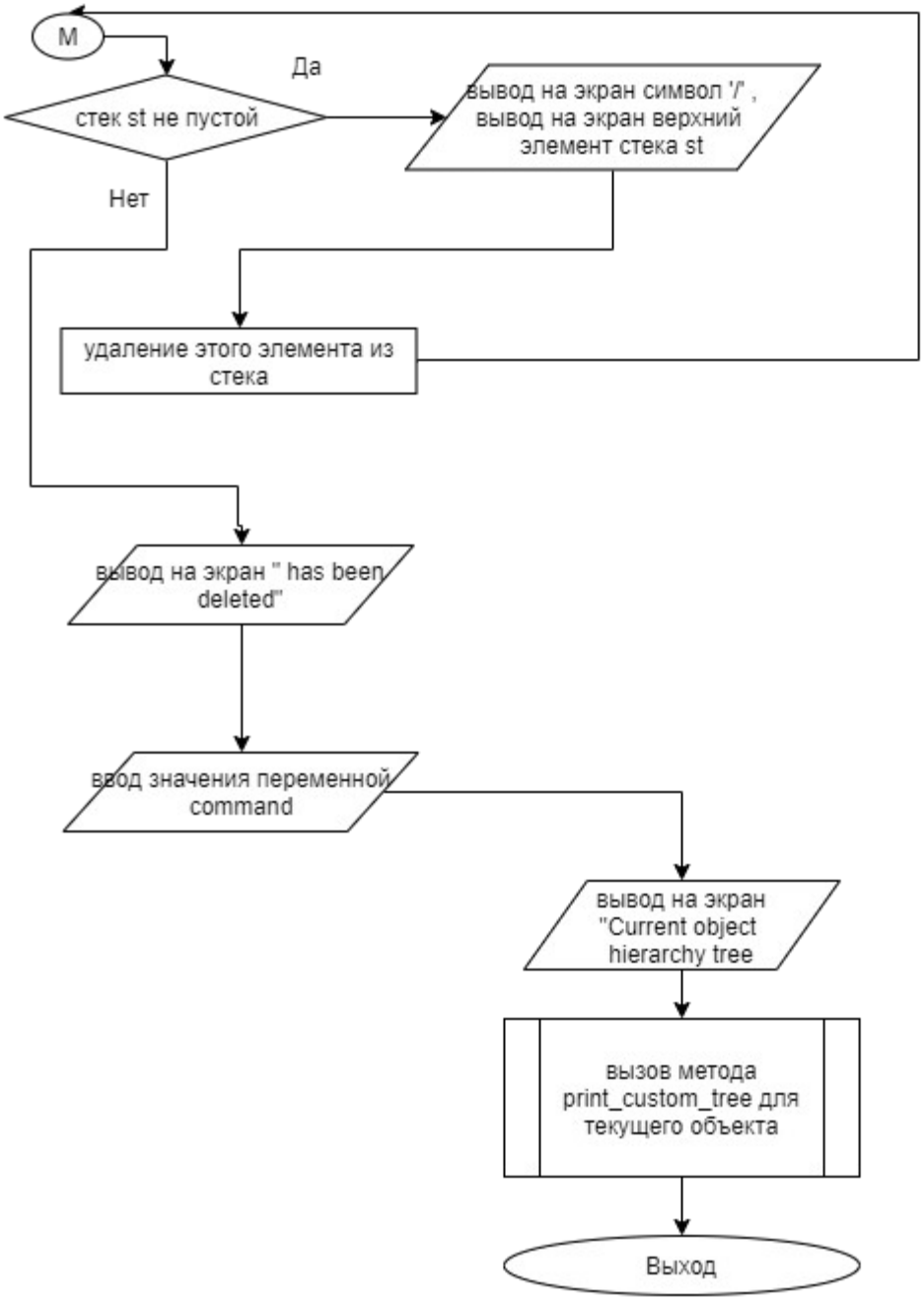

Рисунок 26 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_custom_2.cpp

Листинг 1 – cl_custom_2.cpp

```
#include "cl_custom_2.h"

cl_custom_2::cl_custom_2(cl_custom_base*    p_main,    string    s_name)    :
cl_custom_base(p_main, s_name) {}
//  конструктор  класса  cl_custom_2,  который  вызывает  конструктор  базового
класса cl_custom_base с параметрами
// p_main и s_name
```

5.2 Файл cl_custom_2.h

Листинг 2 – cl_custom_2.h

```
#ifndef __CL_CUSTOM_2__H
#define __CL_CUSTOM_2__H

#include "cl_custom_base.h"

class  cl_custom_2  :  public  cl_custom_base  {  //  определение  класса
cl_custom_2,
                                    // наследуемый от класса cl_custom_base
public:
   cl_custom_2(cl_custom_base* p_main, string s_name);
   // объявление конструктора класса cl_custom_2, принимающий
   // указатель на cl_custom_base и строку s_name
};

#endif
```

5.3 Файл cl_custom_3.cpp

Листинг 3 – cl_custom_3.cpp

```
#include "cl_custom_3.h"

cl_custom_3::cl_custom_3(cl_custom_base*    p_main,    string    s_name)    :
cl_custom_base(p_main, s_name) {}
//  конструктор  класса  cl_custom_3,  который  вызывает  конструктор  базового
класса cl_custom_base с параметрами
// p_main и s_name
```

5.4 Файл cl_custom_3.h

Листинг 4 – cl_custom_3.h

```
#ifndef __CL_CUSTOM_3__H
#define __CL_CUSTOM_3__H

#include "cl_custom_base.h"

class  cl_custom_3  :  public  cl_custom_base  {  //  определение  класса
cl_custom_3,
public:                                 // наследуемый от класса cl_custom_base
   cl_custom_3(cl_custom_base* p_main, string s_name);
   // объявление конструктора класса cl_custom_3, принимающий
   // указатель на cl_custom_base и строку s_name
};

#endif
```

5.5 Файл cl_custom_4.cpp

Листинг 5 – cl_custom_4.cpp

```
#include "cl_custom_4.h"

cl_custom_4::cl_custom_4(cl_custom_base*    p_main,    string    s_name)    :
cl_custom_base(p_main, s_name) {}
//  конструктор  класса  cl_custom_4,  который  вызывает  конструктор  базового
класса cl_custom_base с параметрами
// p_main и s_name
```

5.6 Файл cl_custom_4.h

Листинг 6 – cl_custom_4.h

```
#ifndef __CL_CUSTOM_4__H
#define __CL_CUSTOM_4__H

#include "cl_custom_base.h"

class cl_custom_4 : public cl_custom_base { // определение класса
cl_custom_4,
public:                                 // наследуемый от класса cl_custom_base
   cl_custom_4(cl_custom_base* p_main, string s_name);
   // объявление конструктора класса cl_custom_4, принимающий
   // указатель на cl_custom_base и строку s_name
};

#endif
```

5.7 Файл cl_custom_5.cpp

Листинг 7 – cl_custom_5.cpp

```
#include "cl_custom_5.h"

cl_custom_5::cl_custom_5(cl_custom_base*    p_main,    string    s_name)    :
cl_custom_base(p_main, s_name) {}
// конструктор класса cl_custom_5, который вызывает конструктор базового
класса cl_custom_base с параметрами
// p_main и s_name
```

5.8 Файл cl_custom_5.h

Листинг 8 – cl_custom_5.h

```
#ifndef __CL_CUSTOM_5__H
#define __CL_CUSTOM_5__H

#include "cl_custom_base.h"

class cl_custom_5 : public cl_custom_base { // определение класса
cl_custom_5,
public:                                 // наследуемый от класса cl_custom_base
```

```
   cl_custom_5(cl_custom_base* p_main, string s_name);
   // объявление конструктора класса cl_custom_5, принимающий
   // указатель на cl_custom_base и строку s_name
};

#endif
```

5.9 Файл cl_custom_6.cpp

Листинг 9 – cl_custom_6.cpp

```
#include "cl_custom_6.h"

cl_custom_6::cl_custom_6(cl_custom_base*    p_main,    string    s_name)    :
cl_custom_base(p_main, s_name) {}
//  конструктор  класса  cl_custom_6,  который  вызывает  конструктор  базового
класса cl_custom_base с параметрами
// p_main и s_name
```

5.10 Файл cl_custom_6.h

Листинг 10 – cl_custom_6.h

```
#ifndef __CL_CUSTOM_6__H
#define __CL_CUSTOM_6__H

#include "cl_custom_base.h"

class  cl_custom_6  :  public  cl_custom_base  {  //  определение  класса
cl_custom_6,
public:                              // наследуемый от класса cl_custom_base
   cl_custom_6(cl_custom_base* p_main, string s_name);
   // объявление конструктора класса cl_custom_6, принимающий
   // указатель на cl_custom_base и строку s_name
};

#endif
```

5.11 Файл cl_custom_application.cpp

Листинг 11 – cl_custom_application.cpp

```
#include "cl_custom_application.h"
#include "cl_custom_2.h"
#include "cl_custom_3.h"
#include "cl_custom_4.h"
#include "cl_custom_5.h"
#include "cl_custom_6.h"
#include "stack"

cl_custom_application::cl_custom_application(cl_custom_base*    p_main)    :
cl_custom_base(p_main) {}
// определение конструктора класса

void cl_custom_application::build_tree_objects() {
   // определение метода build_tree_objects
   cout << "Object tree";

   string path_custom, sub_name;
   //  path_custom  -  используется  для  указания  пути  к  объекту  в  дереве
объектов
   // при построении иерархии

   // sub_name - используется для указания номера класса объекта, который
   // будет создан в методе
   int class_number;

   cin >> sub_name;
   this->set_custom_name(sub_name);

   cl_custom_base* parent;
   cl_custom_base* last_created = this;
   cin >> path_custom;

   while(path_custom  !=  "endtree")  {  //  пока  строка  path_custom  не  будет
равна 'endtree'
      cin >> sub_name >> class_number;

      parent = last_created->get_object_path(path_custom);
      //вызов метода get_object_path для объекта last_created,
      //передавая ему строку path_custom, чтобы найти объект
      //указанному в path_custom, полученный объект присваивается переменной
parent

      if(parent == nullptr) {

         this->print_custom_tree();

         cout  <<  endl  <<  "The  head  object  "  <<  path_custom  <<  "  is  not
found";
         exit(1);
      }
```

```
     if(parent->get_sub_custom(sub_name)   !=   nullptr)   {   //   получаем
подчиненный
        // объект с именем sub_name у род объекта parent, проверка что
подчин объект существует
        cout << endl << path_custom << " Dubbing the names of subordinate
objects";
     } else {
        switch(class_number) {
           case 1:
              last_created = new cl_custom_application(parent);
              break;
           case 2:
              last_created = new cl_custom_2(parent, sub_name);
              break;
           case 3:
              last_created = new cl_custom_3(parent, sub_name);
              break;
           case 4:
              last_created = new cl_custom_4(parent, sub_name);
              break;
           case 5:
              last_created = new cl_custom_5(parent, sub_name);
              break;
           case 6:
              last_created = new cl_custom_6(parent, sub_name);
              break;
           default:
              break;
        }
     }
     cin >> path_custom; // тестить
  }
}

int cl_custom_application::exec_custom_app() {
   // логика пользовательского приложения
   string command, input;
   // command используется для хранения команды, вводимых пользователем во
время
   // выполнения алгоритма

   // input используется для хранения доп данных, например: если
   // команда FIND используется для поиска объекта, то input может содержать
   // имя объекта, который нужно найти

   cl_custom_base* current_obj = this;
   cl_custom_base* extra_obj = this;
   stack<string> st;
   // стек st используется для хранения и отслеживания изменений объектов при
выполнении
   // операций в методе exec_custom_app

   this->print_custom_tree();
   cin >> command;
```

```
   while(command != "END") { // пока строка command не равна END

      cin >> input;

      if(command == "SET") { // если строка command равна SET
         extra_obj = current_obj->get_object_path(input);

         if(extra_obj != nullptr) {
            current_obj = extra_obj;
            cout << endl << "Object is set: " << current_obj-
>get_custom_name();
         } else {
            cout << endl << "The object was not found at the specified
coordinate: " << input;
         }

      }
      else if(command == "FIND") { // если строка command равна FIND
         extra_obj = current_obj->get_object_path(input);

         if(extra_obj != nullptr) {
            cout << endl << input << "       Object name: " << extra_obj-
>get_custom_name();
         } else {
            cout << endl << input << "     Object is not found";
         }

      }
      else if(command == "MOVE") { // если строка command равна MOVE
         extra_obj = current_obj->get_object_path(input);

         if(current_obj->set_head_object(extra_obj)) {
            cout << endl << "New head object: " << extra_obj-
>get_custom_name();
         }
         else if(extra_obj == nullptr) {
            cout << endl << input << "      Head object is not found";
         }
         else if(extra_obj->get_sub_custom(current_obj->get_custom_name()) !=
nullptr) {
            cout << endl << input << "      Dubbing the names of subordinate
objects";
         } else {
            cout << endl << input << "        Redefining the head object
failed";
         }
      }
      else if(command == "DELETE") { // если строка command равна DELETE
         extra_obj = current_obj->get_sub_custom(input);
         if(extra_obj != nullptr) {

            while(extra_obj->get_main_custom() != nullptr) {
               st.push(extra_obj->get_custom_name());
               extra_obj = extra_obj->get_main_custom();
```

```
                }

                current_obj->delete_custom_sub_name(input);
                cout << endl << "The object ";
                while(!st.empty()) {
                   cout << '/' << st.top();
                   st.pop();
                }
                cout << " has been deleted";
             }
          }
          cin >> command;
       }

       cout << endl << "Current object hierarchy tree";
       this->print_custom_tree();
       return 0;
    }
```

5.12 Файл cl_custom_application.h

Листинг 12 – cl_custom_application.h

```
    #ifndef __CL_CUSTOM_APPLICATION__H
    #define __CL_CUSTOM_APPLICATION__H

    #include "cl_custom_base.h"

    #include <iostream>

    class cl_custom_application : public cl_custom_base {
       // определение класса cl_custom_application, наследующий от cl_custom_base
    public:
       cl_custom_application(cl_custom_base* p_main_custom_object);
       // конструктор - принимающий указатель на cl_custom_base
       void build_tree_objects();
       // метод - строит древовидную структуру объектов
       int exec_custom_app();
       // метод - сборка приложения
    };
    #endif
```

5.13 Файл cl_custom_base.cpp

Листинг 13 – cl_custom_base.cpp

```
#include "cl_custom_base.h"
#include "stack"

using namespace std;

cl_custom_base::cl_custom_base(cl_custom_base* p_main, string s_name) {
   this->p_main_custom_object = p_main;
   this->custom_name = s_name;

   if(p_main_custom_object != nullptr) {
      p_main_custom_object->p_sub_customs.push_back(this);
   }
   // добавление текущего объекта в список подчиненных основного объекта,
если
   // основной объект сущесвует
};

cl_custom_base::~cl_custom_base() {
   // освобождение памяти, выделенной для подчиненных объектов
   for(int i = 0; i < p_sub_customs.size(); i++) {
      delete p_sub_customs[i];
   }
};

cl_custom_base* cl_custom_base::get_main_custom() {
   return this->p_main_custom_object;
};

bool cl_custom_base::set_custom_name(string new_custom_name) {
   if(p_main_custom_object != nullptr) {
      for(int i = 0; i < p_main_custom_object -> p_sub_customs.size(); i++) {
         // есть ли объект с таким же именем среди подчиненных объектов
основного
         // объекта - если такой объект найден, возврат false, - не
получилось
         // установить новое имя
         if(p_main_custom_object->p_sub_customs[i]->get_custom_name()      ==
new_custom_name) {
            return false;
         }
         //присваивание нового имени полю custom_name текущего объекта и
возврат true,
         // если имя не дублируется среди подчиненных объектов основного
объекта
      }
   }

   this->custom_name = new_custom_name;
   return true;
};
```

```
string cl_custom_base::get_custom_name() {
   return this->custom_name;
}

cl_custom_base* cl_custom_base::get_sub_custom(string s_name) {

   for(int i = 0; i < p_sub_customs.size(); i++) {
      if(p_sub_customs[i]->get_custom_name() == s_name) {
         return p_sub_customs[i];
      }
   }
   // поиск подчиненного объекта по имени s_name
   // перебор всех подчиненных объектов текущего объекта 'p_sub_customs'
   // совпало ли имя подчиненного объекта с искомым именем s_name, если да
   // возврат указатель на этот подчиненный объект

   return nullptr;
};

cl_custom_base* cl_custom_base::search_tree(string s_name) {
   // поиск объекта с заданным именем s_name в дереве объектов, используя
поиск в ширину

   // создание очереди q для обхода дерева в ширину

   // цикл while- до тех пор пока очередь не станет пустой

   // поиск объекта s_name в каждом уровне дерева и добавление его в очередь

   // после нахождения 1-ого объекта с искомым именем, поиск останавливается
-

   // возврат результата found
   queue<cl_custom_base*> q;
   cl_custom_base* found = nullptr;
   q.push(this);
   while(!q.empty()){
      cl_custom_base* e = q.front();
      if(e->get_custom_name() == s_name) {
         if(found != nullptr) {
            return nullptr;
         } else {
            found = e;
         }
      }

      for(int i = 0; i < e->p_sub_customs.size(); i++) {
         q.push(e->p_sub_customs[i]);
      }

      q.pop();
   }

   return found;
```

```
}

cl_custom_base* cl_custom_base::search_current(string s_name) {
   // рекурсивный поиск объекта с заданным именем
   if(p_main_custom_object != nullptr) {
      return p_main_custom_object->search_current(s_name);
      // вызов метода search_current для продолжения поиска вверх по иерархии
   } else {
      return search_tree(s_name);
      // поиск в глубину в текущем поддереве
   }
}

void cl_custom_base::print_custom_tree(int layer) {
   // печать дерева объектов, layer - используется для отступа при выводе
   // имени объектов. выводится имя текущего объекта с отступом, затем
   // рекурсивно вызывается метод для каждого подчиненного объекта
   cout << endl;

   for(int i = 0; i < layer; i++) {
      cout << "    ";
   }

   cout << this->get_custom_name();
   for(int i = 0; i < p_sub_customs.size(); i++) {
      p_sub_customs[i]->print_custom_tree(layer + 1);
   }
}

void cl_custom_base::print_tree_two(int layer) {
   // вывод имени текущего объекта и сообщение о его готовности(is ready/ is
not ready)

   //  метод  рекурсивно  вызывается  для  каждого  подчиненного  объекта  с
увеличением
   // отступа(i_space + 1)
   cout << endl;

   for(int i = 0; i < layer; ++i) {
      cout << "    ";
   }

   if(this->i_space != 0) {
      cout << this->get_custom_name() << " is ready";
   } else {
      cout << this->get_custom_name() << " is not ready";
   }

   for(int i = 0; i < p_sub_customs.size(); ++i) {
      p_sub_customs[i]->print_tree_two(i_space + 1);
   }
}

void cl_custom_base::set_state(int i_space) {
   if(p_main_custom_object == nullptr || p_main_custom_object->i_space!= 0) {
```

```
            this->i_space = i_space;
        }

        if(i_space == 0) {
            this->i_space = i_space;
            for(int i = 0; i < p_sub_customs.size(); i++) {
                p_sub_customs[i]->set_state(i_space);
            }
        }
    }

    bool cl_custom_base::set_head_object(cl_custom_base* new_p_head_object) {
        if(this->get_main_custom() == new_p_head_object) {
        // проверка - является ли новый главный объект уже текущим главным
    объектом
            return true;
        }
        if(this->get_main_custom() == nullptr || new_p_head_object == nullptr) {
        // проверка - существует ли уже подчиненный объект у нового главного
    объекта
        // с таким же именем , как у текущего объекта
            return false;
        }
        if(new_p_head_object->get_sub_custom(this->get_custom_name()) != nullptr)
    {
            return false;
        // добавление текущего объекта в список подчиненных объектов нового
    главного
        // объекта
        }

        // обход дерева объектов начиная с текущего объекта и двигаясь по его
    подчиненным
        // объектам в глубину, если находится новый главный объект - возврат false
        //
        // поиск и удаление текущего объекта из старого главного объекта, после
    завершения
        // цикла и обхода дерева объектов, текущий объект удаляется из списка
    подчиненных
        // объектов его старого главного объекта
        //
        // после удаления из старого главного объекта, текущий объект добавляется
    в список
        // подчиненных объектов нового главного объекта
        stack<cl_custom_base*> st;
        st.push(this);
        while(!st.empty()){
            cl_custom_base* current = st.top();
            st.pop();
            if(current == new_p_head_object) {
                return false;
            }

            for(int i = 0; i < current->p_sub_customs.size(); i++) {
                st.push(current->p_sub_customs[i]);
```

```
            }
        }

        vector<cl_custom_base*> & v = this->get_main_custom()->p_sub_customs;
        for(int i = 0; i < v.size(); i++) {
            if(v[i]->get_custom_name() == this->get_custom_name()) {
                v.erase(v.begin() + i);
                new_p_head_object->p_sub_customs.push_back(this);
                return true;
            }
        }
        return false;
    }

    void cl_custom_base::delete_custom_sub_name(string sub_name) {
        // метод перебирает список подчиненных объектов, ища объект с заданным
    именем sub_name
        // как только объект найден,он удаляется из вектора и освобождается память
        vector<cl_custom_base*> & v = this->p_sub_customs;

        for(int i = 0; i < v.size(); i++) {

            if(v[i]->get_custom_name()==sub_name) {
                delete v[i];
                v.erase(v.begin() + i);
                return;
            }
        }
    }

    cl_custom_base* cl_custom_base::get_object_path(string path_custom) {
        // проверка на пустоту строки - если строка пуста возврат nullptr - нечего
    искать
        // если строк path_custom равна '.' - возврат текущего объекта

        // если строк начинается с '//' - поиск объекта в текущем поддереве -
    игнорируя
        // уровни выше, путем вызова search_current

        if(path_custom.empty()) {
            return nullptr;
        }

        if(path_custom == "."){
            return this;
        }

        if(path_custom[0]=='.') {
            return search_tree(path_custom.substr(1));
        }

        if(path_custom.substr(0, 2) == "//") {
            return this->search_current(path_custom.substr(2));
        }
```

```
   if(path_custom[0] != '/') {
      size_t slash = path_custom.find('/');
      cl_custom_base*  sub_ptr  =  this->get_sub_custom(path_custom.substr(0,
slash));
      if(sub_ptr == nullptr || slash == string::npos) {
         return sub_ptr;
      }

      return sub_ptr->get_object_path(path_custom.substr(slash + 1));
   }

   cl_custom_base* root = this;
   while(root->get_main_custom() != nullptr) {
      root = root->get_main_custom();
   }

   if(path_custom == "/") {
      return root;
   }

   return root->get_object_path(path_custom.substr(1));
}
```

5.14 Файл cl_custom_base.h

Листинг 14 – cl_custom_base.h

```
#ifndef __CL_CUSTOM_BASE__H
#define __CL_CUSTOM_BASE__H

#include <iostream>
#include <vector>
#include <queue>
#include <string>

using namespace std;

class cl_custom_base {
private:
   int i_space = 0; // приватное поле, хранение отступа
   string custom_name; // поле, хранение имени объекта
   cl_custom_base* p_main_custom_object; // поле, хранение указателя на глав
объект
   vector<cl_custom_base*>  p_sub_customs;  //  поле,  хранение  указателей  на
подчиненные объекты
public:
   cl_custom_base(cl_custom_base* p_main, string s_name = "Base Object");
   // конструктор с параметрами
   ~cl_custom_base(); // деструктор
   bool set_custom_name(string new_custom_name); // установка имени объекта
```

```
   string get_custom_name(); // получение имени объекта
   cl_custom_base* get_main_custom(); // получение главного объекта
   cl_custom_base*   get_sub_custom(string   custom_name);   //   получение
подчиненного объекта по имени
   cl_custom_base* search_tree(string custom_name); // поиск объекта в дереве
   cl_custom_base*  search_current(string  custom_name);  //  поиск  объекта  в
текущем уровне дерева
   void print_custom_tree(int layer = 0); // печать дерева объектов
   void  print_tree_two(int  layer  =  0);  //  печать  дерева  объектов  с  двумя
пробелами(отступ)
   void set_state(int layer); // установка состояния объекта
   bool  set_head_object(cl_custom_base*  p_main_custom_object);  //  установка
главного объекта
   void  delete_custom_sub_name(string  sub_name);  //  удаление  подчиненного
объекта по имени
   cl_custom_base* get_object_path(string path_custom); // получение объекта
по пути
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_custom_application.h"

#include <string>
#include <iostream>

int main() {
   cl_custom_application   obj_custom_app(nullptr);   //   создание   объекта
obj_custom_app
   obj_custom_app.build_tree_objects(); // вызов метода для построения дерева
объектов
   return(obj_custom_app.exec_custom_app());  //  выполнение  приложения  и
возврат его результата
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 23.

Таблица 23 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

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
	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 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

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_3 object_3 object_7	object_3 object_3 object_7

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).