Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

```
«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»
```

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Поиск головного объекта выполняется от последнего созданного объекта. Первоначально последним созданным объектом считается корневой объект. Если для головного объекта обнаруживается дубляж имени в непосредственно подчиненных объектах, то объект не создается. Если обнаруживается дубляж имени на дереве иерархии объектов, то объект не создается. Если номер класса объекта задан некорректно, то объект не создается.

Вывод иерархического дерева объектов на консоль.

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:

- метод поиска объекта на ветке дереве иерархии от текущего по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на искомой ветке дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод вывода иерархии объектов (дерева или ветки) от текущего объекта (допускается использовать один целочисленный параметр со значением по-умолчанию);
- метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта (допускается использовать один целочисленный параметр со значением по-умолчанию);
- метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Устаревший метод вывода из задачи КВ_1 убрать.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется. При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

1. Построить дерево объектов системы (в методе корневого объекта построения исходного дерева объектов).

2. В методе корневого объекта запуска моделируемой системы реализовать:2.1. Вывод на консоль иерархического дерева объектов в следующем виде:

```
root
    ob_1
        ob_2
    ob_3
        ob_4
            ob_5
        ob_6
            ob_7
```

где: root - наименование корневого объекта (приложения).

2.2. Переключение готовности объектов согласно входным данным (командам). 2.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

```
root  is ready
    ob_1  is ready
        ob_2  is ready
    ob_3  is ready
        ob_4 is not ready
            ob_5 is not ready
        ob_6 is ready
            ob_7 is not ready
```

1.1 Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Последовательность ввода организовано так, что головной объект для очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка:

```
«Наименование корневого объекта»
```

Со второй строки:

```
«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»
. . . . .
endtree
```

Со следующей строки вводятся команды включения или отключения объектов

```
«Наименование объекта» «Номер состояния объекта»
```

Пример ввода:

```
app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
```

1.2 Описание выходных данных

Вывести иерархию объектов в следующем виде:

```
Object tree
«Наименование корневого объекта»
    «Наименование объекта 1»
        «Наименование объекта 2»
    «Наименование объекта 3»
. . . . .
The tree of objects and their readiness
«Наименование корневого объекта» «Отметка готовности»
    «Наименование объекта 1» «Отметка готовности»
        «Наименование объекта 2» «Отметка готовности»
    «Наименование объекта 3» «Отметка готовности»
. . . . .
«Отметка готовности» - равно «is ready» или «is not ready»
```

Отступ каждого уровня иерархии 4 позиции.

Пример вывода:

```
Object tree
app_root
    object_01
        object_07
```

```
    object_02
        object_04
        object_05
The tree of objects and their readiness
app_root is ready
    object_01 is ready
        object_07 is not ready
    object_02 is ready
        object_04 is ready
        object_05 is not ready
```

2 МЕТОД РЕШЕНИЯ

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 p_sub_customs класса cl_custom_base предназначен для хранение подчиненных объектов;
- объект класса cl_2, cl_3, cl_4, cl_5, cl_6 предназначен для формирования иерархической структуры в дереве 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 — custom_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 set_state — установка значения состояния;
  - метод cl_custom_base — конструктор;
  - метод ~cl_custom_base — деструктор;
  - метод get_custom_name — получение имени текущего пользовательского объекта, возврат строки, представляющую имя объекта;
  - метод set_custom_name — установка имени текущего пользовательского объекта;
  - метод get_main_custom — получение указателя на родительский объект текущего пользовательского объекта;
  - метод get_sub_custom — получение указателя на подпользовательский объект текущего пользовательского объекта;
  - метод print_custom_tree — вывод информации о пользовательских объектах и их иерархии на экран;
  - метод search_tree — поиск объекта в дереве;
  - метод search_current — поиск объекта начиная от текущего объекта;
  - метод print_tree_two — вывод информации о пользовательских объектах и их иерархии на экран.

Класс cl_custom_application:

- функционал:
  - метод build_tree_objects — создание иерархии объектов;
  - метод exec_app — метод сборки 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 иерархиией	
		cl_custom_application	public		2
		cl_custom_2	public		3
		cl_custom_3	public		4
		cl_custom_4	public		5
		cl_custom_5	public		6

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
		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 'cl_custom_base', собственные свойства отсутсвуют	
4	cl_custom_3			создание типа объекта, наследуя 'cl_custom_base', собственные свойства отсутсвуют	
5	cl_custom_4			создание типа объекта, наследуя 'cl_custom_base', собственные свойства отсутсвуют	
6	cl_custom_5			создание типа объекта, наследуя 'cl_custom_base', собственные свойства отсутсвуют	
7	cl_custom_6			создание типа объекта, наследуя 'cl_custom_base', собственные свойства отсутсвуют	

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 set_state класса cl_custom_base

Функционал: установка значения состояния.

Параметры: i_state - целочисленный параметр, представляет состояние объекта.

Возвращаемое значение: отсутсвует.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_state класса cl_custom_base

№	Предикат	Действия	№ перехода
1	равен ли указатель 'p_main_custom_object' нулю или 'i_state' у 'p_main_custom_object' не равен 0?	значение переменной 'i_state' присваивается полю 'i_state' текущего объекта	2
			2
2	значение переменной 'i_state' равно 0?	значение переменной 'i_state' присваивается полю 'i_state' текущего объекта	3
			∅
3		объявление целочисленной переменной i = 0	4
4	переменная 'i' меньше размера ветора 'p_sub_customs'?	i++; вызов метода 'set_state' для каждого элемента вектора 'p_sub_customs' передавая в качестве аргумента значение 'i_state'	∅
			∅

3.2 Алгоритм конструктора класса cl_custom_base

Функционал: инициализация объектов класса 'cl_custom_base'.

Параметры: p_main - указатель на объект типа 'cl_custom_base', 'custom_name' - строка, имя текущего объекта.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса cl_custom_base

№	Предикат	Действия	№ перехода
1		присваивание значение переменной 'custom_name' переданной в конструктор члену данных 'custom_name', текущего объекта класса 'cl_custom_base', устанавливается имя текущего объекта	2
2		присваивание значения переменной 'p_main_custom_object', которая является членом данных текущего объекта класса 'cl_custom_base', значение этой переменной устанавливается равным значению, переданному в конструктор через параметр 'p_main'	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'	∅
			∅

3.3 Алгоритм деструктора класса cl_custom_base

Функционал: деструктор.

Параметры: отсутствуют.

Алгоритм деструктора представлен в таблице 4.

Таблица 4 – 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	∅
			∅

3.4 Алгоритм метода get_custom_name класса cl_custom_base

Функционал: возврат строки, содержащую имя текущего объекта.

Параметры: отсутствуют.

Возвращаемое значение: имя текущего объекта типа 'cl_custom_base'.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_custom_name класса cl_custom_base

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

3.5 Алгоритм метода set_custom_name класса cl_custom_base

Функционал: установка нового значения имени объекта.

Параметры: new_custom_name - новое имя объекта.

Возвращаемое значение: булевое значение - false/true.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода set_custom_name класса cl_custom_base

№	Предикат	Действия	№ перехода
1	существует ли главный объект 'p_main_custom_object' и	возвращение 'false', что означает, что подообъект с таким именем уже существует в структуре	2

№	Предикат	Действия	№ перехода
	сущетсвует ли подообъект с заданным новым именем 'new_custom_name'?		
			2
2		установка нового имени 'new_custom_name' для объекта	3
3		возврат 'true' - успешное выполнение операции установки нового имени	∅

3.6 Алгоритм метода get_main_custom класса cl_custom_base

Функционал: возврат указателя на пользовательский объект.

Параметры: отсутствуют.

Возвращаемое значение: p_main_custom_object - указатель на главный пользовательский объект.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_main_custom класса cl_custom_base

№	Предикат	Действия	№ перехода
1		возврат указателя 'p_main_custom_object '	∅

3.7 Алгоритм метода get_sub_custom класса cl_custom_base

Функционал: поиск подобъекта с заданным именем в списке подобъектов 'p_sub_customs'.

Параметры: строка 'custom_name' - имя подообъекта, который нужно найти.

Возвращаемое значение: указатель на найденный подобъект.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода get_sub_custom класса cl_custom_base

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

3.8 Алгоритм метода print_custom_tree класса cl_custom_base

Функционал: печать дерева пользовательских объектов.

Параметры: i_space - количество пробелов для форматирования отступа.

Возвращаемое значение: отсутсвует.

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 меньше i_space?	++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 места в векторе	∅

№	Предикат	Действия	№ перехода
		'p_sub_customs', с увеличенным на 1 значением 'i_space' в качестве аргумента	
			∅

3.9 Алгоритм метода search_tree класса cl_custom_base

Функционал: поиск объекта в дереве.

Параметры: custom_name - имя объекта, который нужно найти.

Возвращаемое значение: указатель на объект с именем 'custom_name', если он найден, или 'nullptr' если не найден.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода search_tree класса cl_custom_base

№	Предикат	Действия	№ перехода
1	главный объект p_main_custom_object не равен nullptr	рекурсивно возвращаем метод search_tree с параметром custom_name	∅
		рекурсивно возвращаем метод search_current с параметром custom_name	∅

3.10 Алгоритм метода search_current класса cl_custom_base

Функционал: поиск объекта начиная от текущего объекта.

Параметры: custom_name - имя объекта, который нужно найти.

Возвращаемое значение: указатель на объект который был найден, если найдено несколько объектов с одинаковым именем, возврат nullptr.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода search_current класса cl_custom_base

№	Предикат	Действия	№ перехода
1		создание указателя 'p_found' и инициализация значением 'nullptr'	2
2		создание пустой очереди 'q', которая будет хранить указатели на объекты типа 'cl_custom_base'	3
3		добавление текущего объекта в очередь 'q'	4
4	очередь не пустая	извлекаем объект и сохраняем его во временный указатель p_front	5
			5
5		удаление объекта из очереди	6
6	имя текущего объекта совпадает с искомым именем custom_name		7
	p_found не равно nullptr	возвращаем nullptr	8
		присваиваение переменной 'p_found' значения 'p_front'	7
7		инициализация целочисленной переменной i = 0	8
8	i меньше размера вектора подчиненных объектов	добавление всех подобъектов в очередь	9
			∅
9		возврат p_found	∅

3.11 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 12.

Таблица 12 – 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 < i_space	вывод на экран 4 пробелов	3
			4
4	i_state не равно 0	вывод имени объекта, вывод строки 'is_ready'	5
		вывод имени объекта, вывод строки 'is_not_ready'	5
5	i меньше размера p_sub_customs	рекурсивно вызов метода print_tree_two для каждого подчиненного объекта, увеличение параметра i_space на 1	5
			∅

3.12 Алгоритм метода build_tree_objects класса cl_custom_application

Функционал: создание иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода build_tree_objects класса cl_custom_application

№	Предикат	Действия	№ перехода
1		инициализация строковых переменных main_custom_name, sub_custom_name	2
2		инициализация указателя 'p_main' значением указателя 'this' - который указывает на текущий объект	3

№	Предикат	Действия	№ перехода
3		объявление указателя 'p_sub' и инициализация его значением 'nullptr' - нулевым указателем	4
4		ввод значение переменной main_custom_name	5
5		вызов функции 'set_custom_name' с передачей ей аргумента 'main_custom_name'	6
6		инициализация целочисленной переменной i_class	7
7	истинно	ввод значение переменной main_custom_name	7
			8
8	головной объект main_custom_name равен строки 'endtree'	прерывание работы цикла	18
			9
9		ввод значения в переменные sub_custom_name, i_class	10
10		поиск объекта с именем 'main_custom_name' в дереве объектов и устанавливание его в качестве главного объекта 'p_main'	11
11	'p_main' не равен 'nullptr' и объект с именем 'sub_custom_name' не найден в поддереве объектов		12
			12
12	p_main равно nullptr	продолжение работы цикла	12
			13
13	значение i_class равно 2	создание нового объекта класса 'cl_custom_2', передавая ему указатель на родительский объект 'p_main' и имя 'sub_custom_name', затем присвоение указателя 'p_sub' созданному объекту	18
			14
14	значение i_class равно 3	создание нового объекта класса 'cl_custom_3',	18

№	Предикат	Действия	№ перехода
		передавая ему указатель на родительский объект 'p_main' и имя 'sub_custom_name', затем присвоение указателя 'p_sub' созданному объекту	
			15
15	значение i_class равно 4	создание нового объекта класса 'cl_custom_4', передавая ему указатель на родительский объект 'p_main' и имя 'sub_custom_name', затем присвоение указателя 'p_sub' созданному объекту	18
			16
16	значение i_class равно 5	создание нового объекта класса 'cl_custom_5', передавая ему указатель на родительский объект 'p_main' и имя 'sub_custom_name', затем присвоение указателя 'p_sub' созданному объекту	18
			17
17	значение i_class равно 6	создание нового объекта класса 'cl_custom_6', передавая ему указатель на родительский объект 'p_main' и имя 'sub_custom_name', затем присвоение указателя 'p_sub' созданному объекту	18
			18
18			∅

3.13 Алгоритм метода exec_app класса cl_custom_application

Функционал: метод сборки приложения.

Параметры: отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения работы алгоритма.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода exec_app класса cl_custom_application

№	Предикат	Действия	№ перехода
1		вывод на экран 'Object tree'	2
2		вызов метода 'print_custom_tree'	3
3		инициализация строковой переменной object_name	4
4		инициализация челочисленной переменной i_state	5
5	ввод в object_name допустим	ввод значения в переменную i_state	6
			7
6	существует ли объект с именем object_name	вызов метода set_state к объекту вызванного методом search_tree	6
			8
7		вывод на экран "The tree of objects and their readiness"	8
8		вызов метода 'print_tree_two()'	∅

3.14 Алгоритм конструктора класса cl_custom_application

Функционал: конструктор.

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

Алгоритм конструктора представлен в таблице 15.

Таблица 15 – 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	∅

3.15 Алгоритм конструктора класса cl_custom_2

Функционал: конструктор.

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

Алгоритм конструктора представлен в таблице 16.

Таблица 16 – Алгоритм конструктора класса cl_custom_2

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_2', который вызывает конструктор базового класса 'cl_custom_base' с передачей ему указателя на главный объект 'p_main' и имя объекта 'custom_name'	∅

3.16 Алгоритм конструктора класса cl_custom_4

Функционал: конструктор.

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

Алгоритм конструктора представлен в таблице 17.

Таблица 17 – 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 'custom_name'	∅

3.17 Алгоритм конструктора класса cl_custom_5

Функционал: конструктор.

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

Алгоритм конструктора представлен в таблице 18.

Таблица 18 – 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 'custom_name'	∅

3.18 Алгоритм конструктора класса cl_custom_6

Функционал: конструктор.

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

Алгоритм конструктора представлен в таблице 19.

Таблица 19 – 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 'custom_name'	∅

3.19 Алгоритм конструктора класса cl_custom_3

Функционал: конструктор.

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

Алгоритм конструктора представлен в таблице 20.

Таблица 20 – 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	∅

№	Предикат	Действия	№ перехода
		указателя на главный объект 'p_main' и имя объекта 'custom_name'	

3.20 Алгоритм функции main

Функционал: основной алгоритм работы программы.

Параметры: отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения работы программы.

Алгоритм функции представлен в таблице 21.

Таблица 21 – 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 1-14.

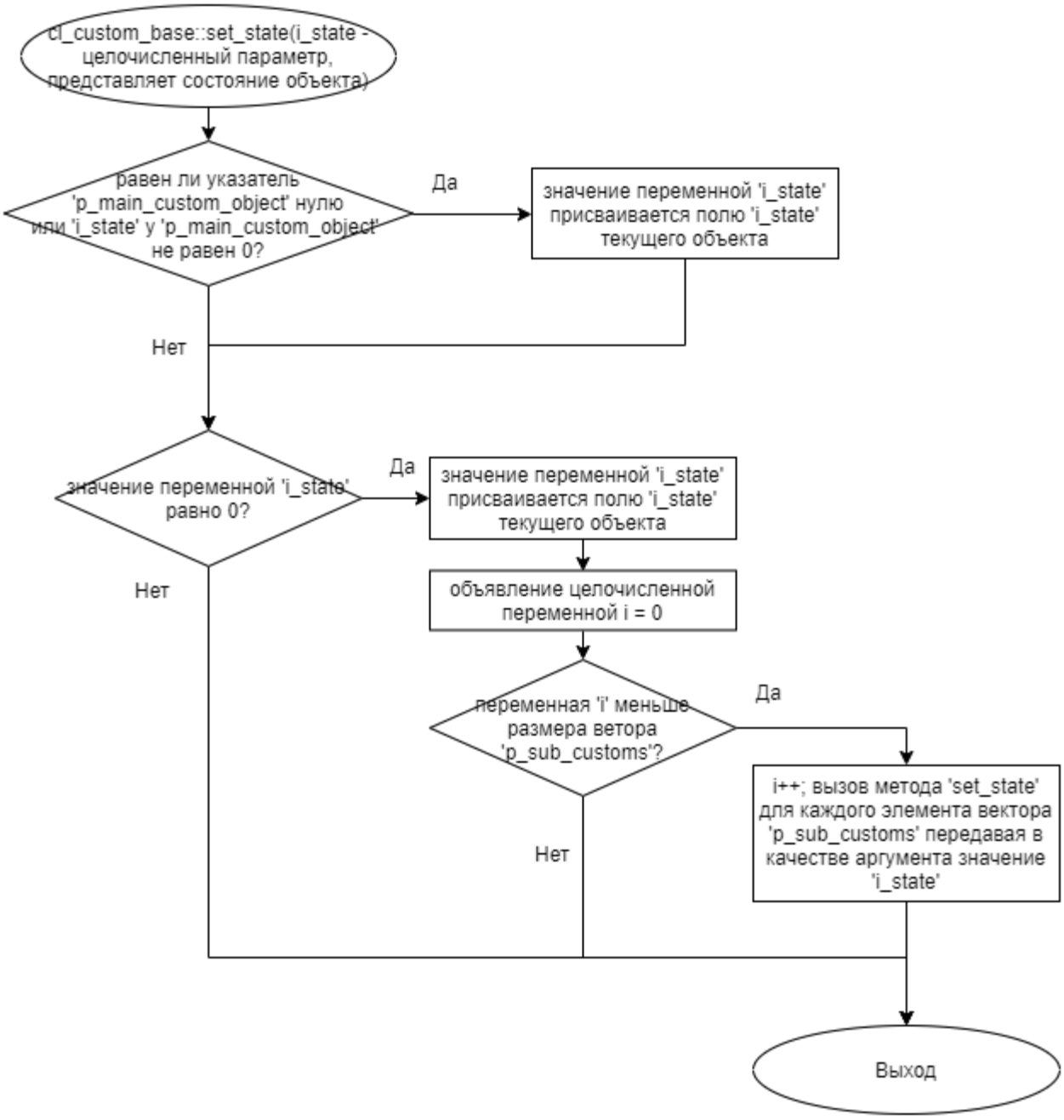

Рисунок 1 – Блок-схема алгоритма

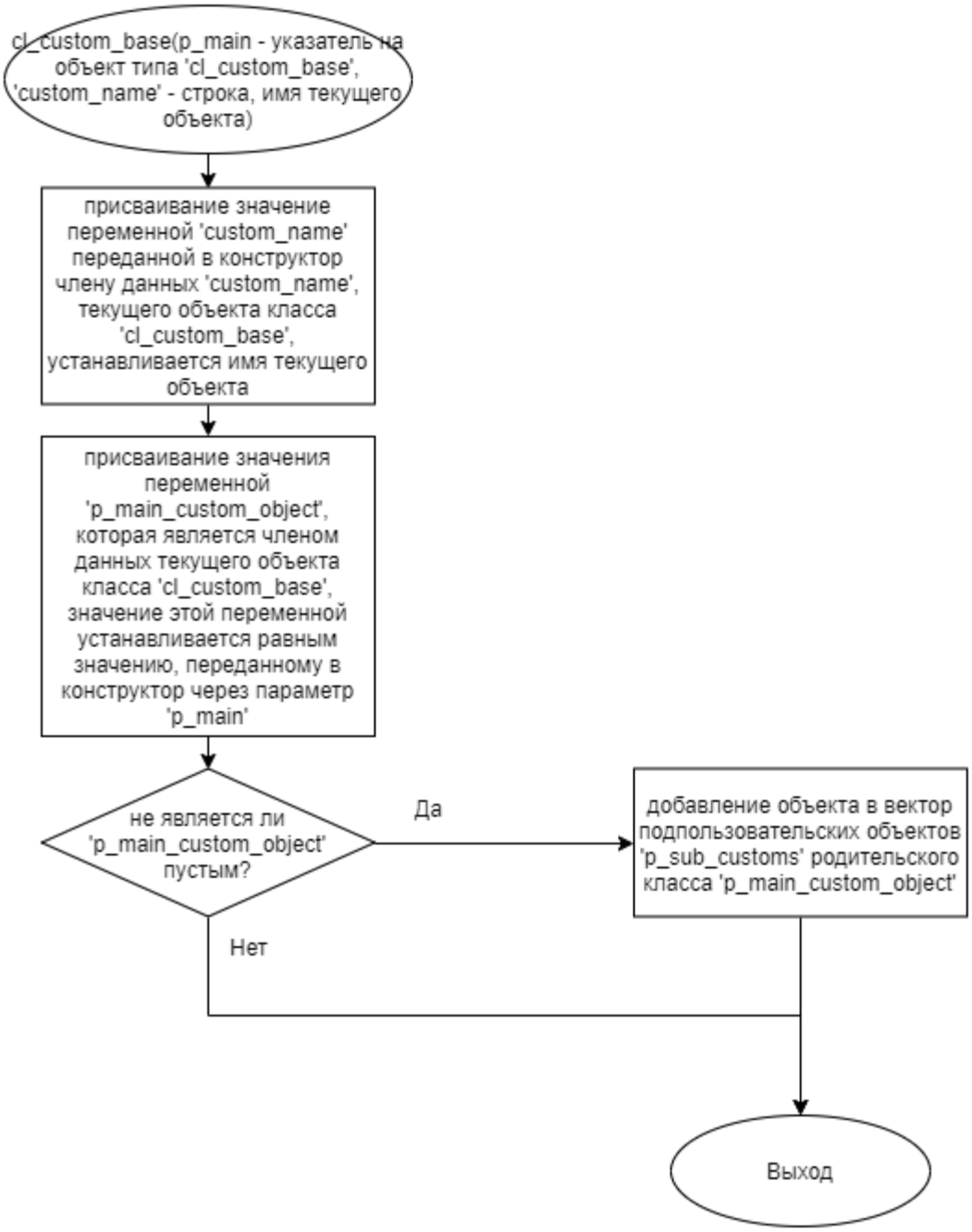

Рисунок 2 – Блок-схема алгоритма

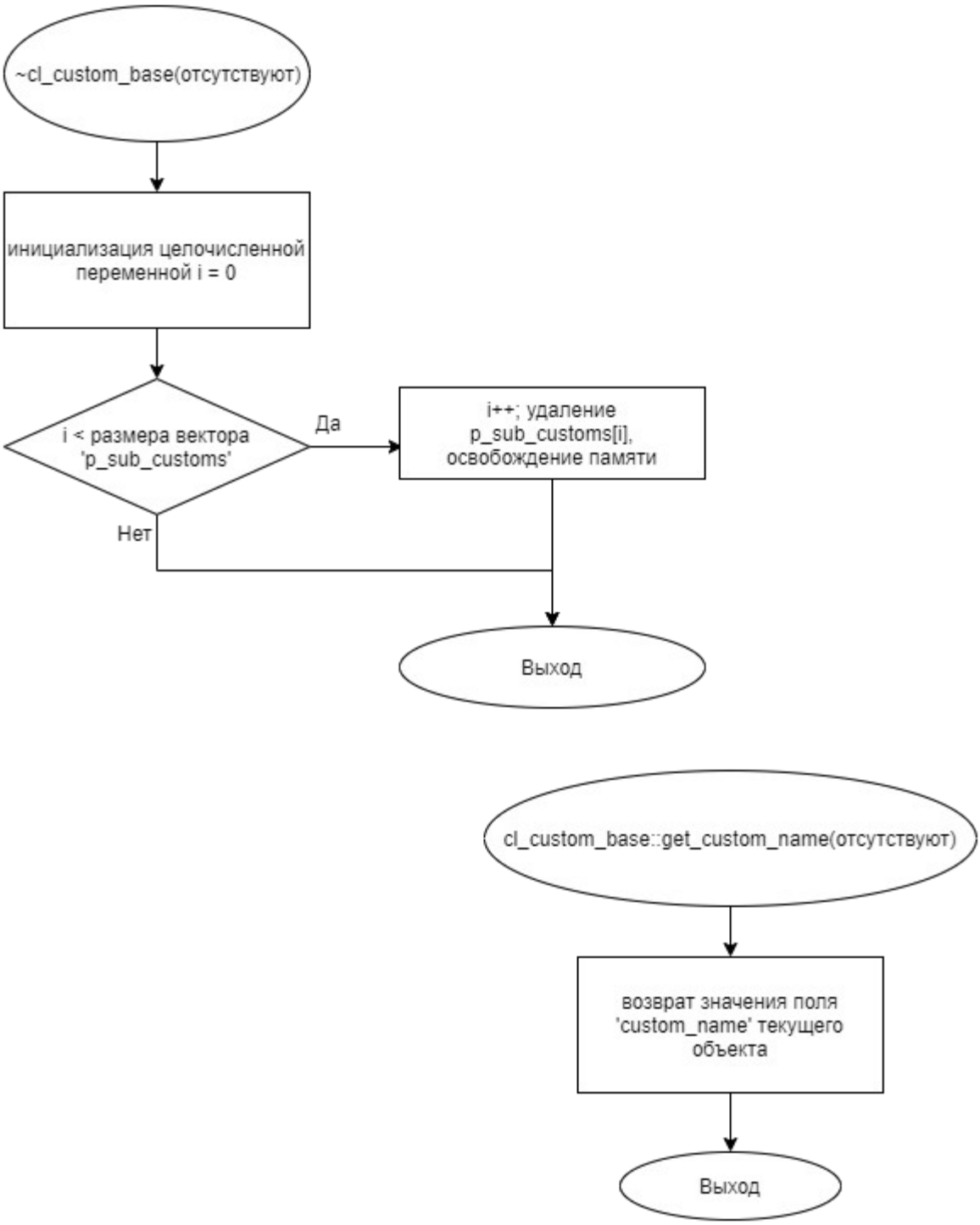

Рисунок 3 – Блок-схема алгоритма

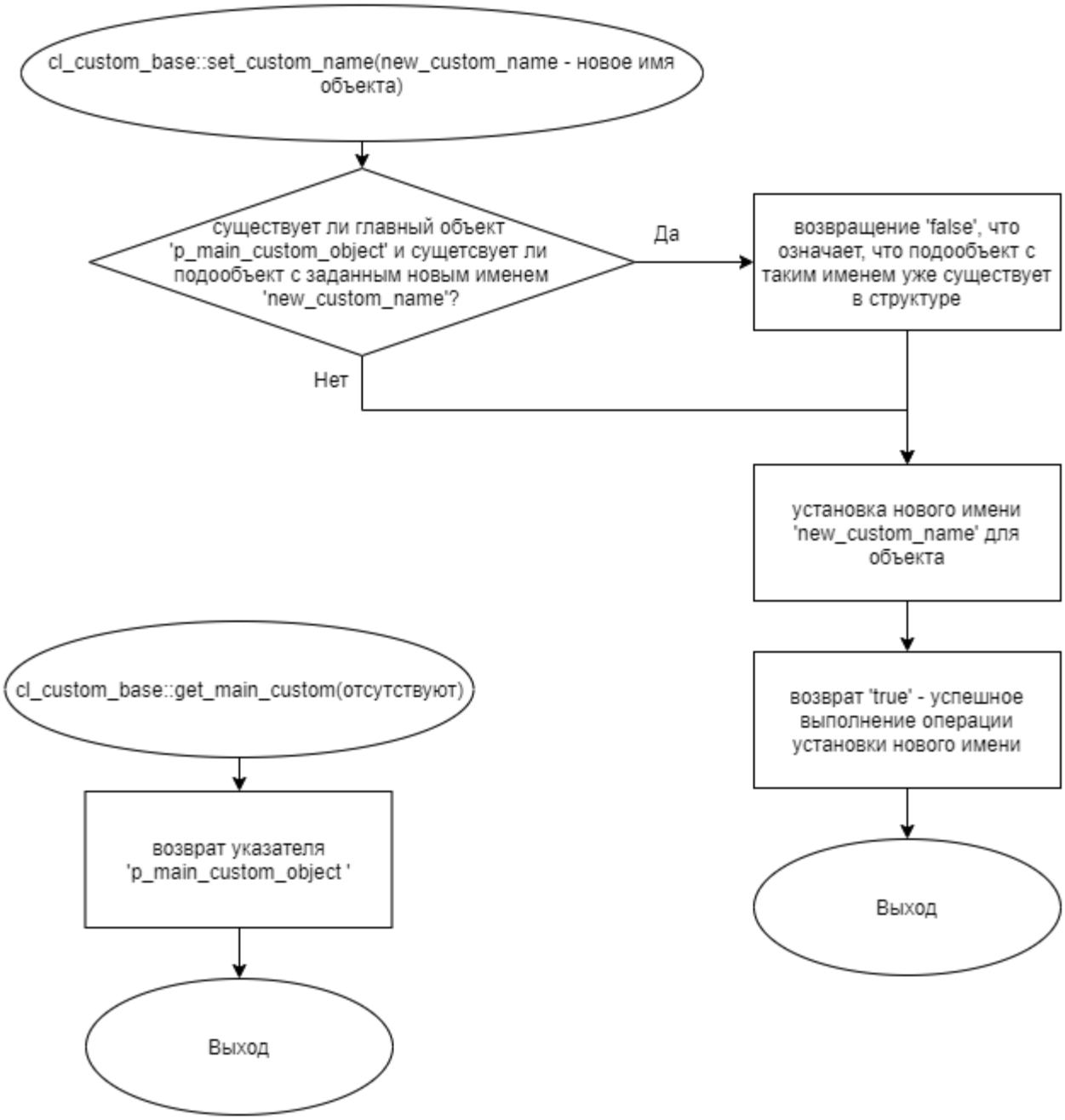

Рисунок 4 – Блок-схема алгоритма

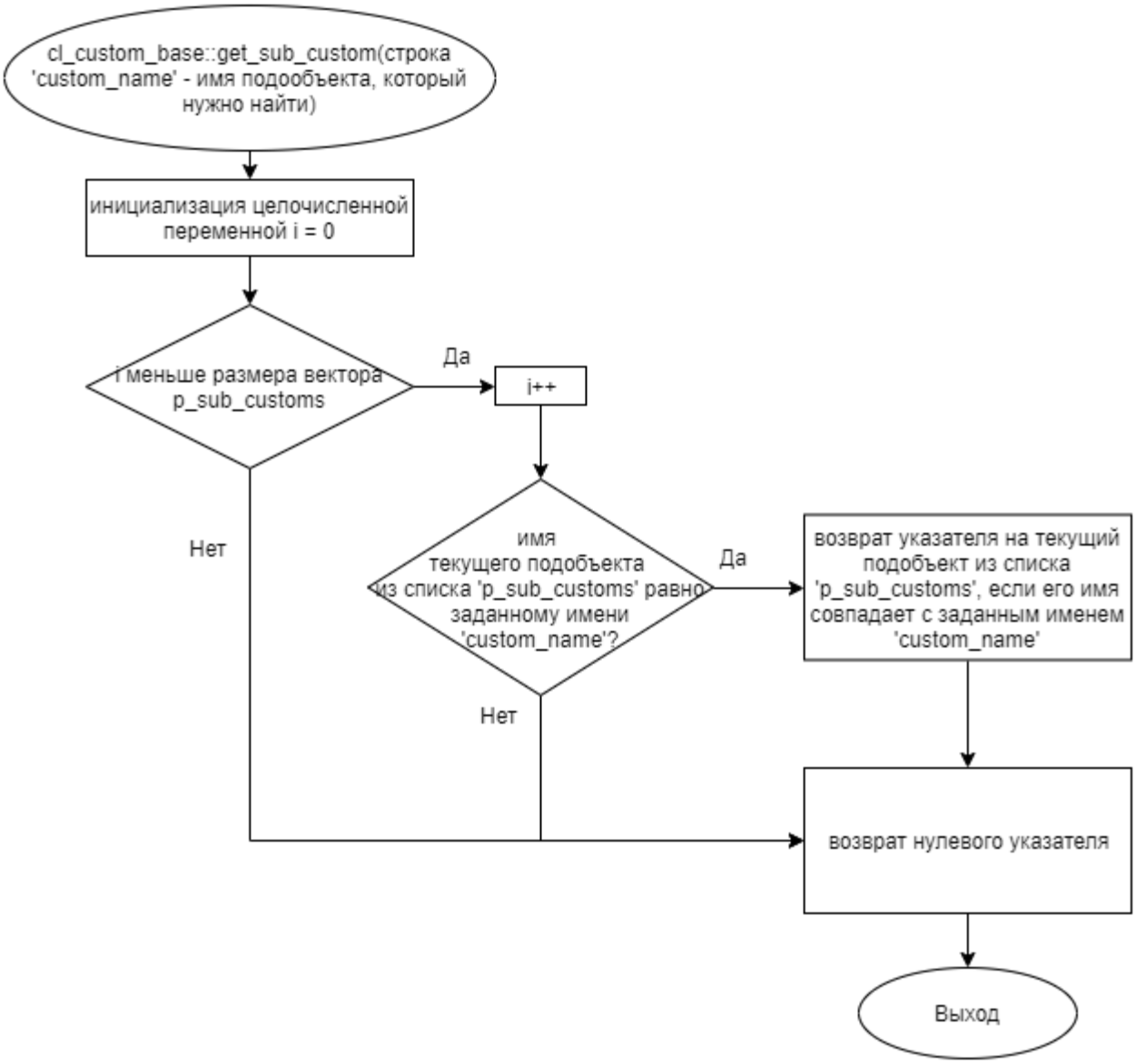

Рисунок 5 – Блок-схема алгоритма

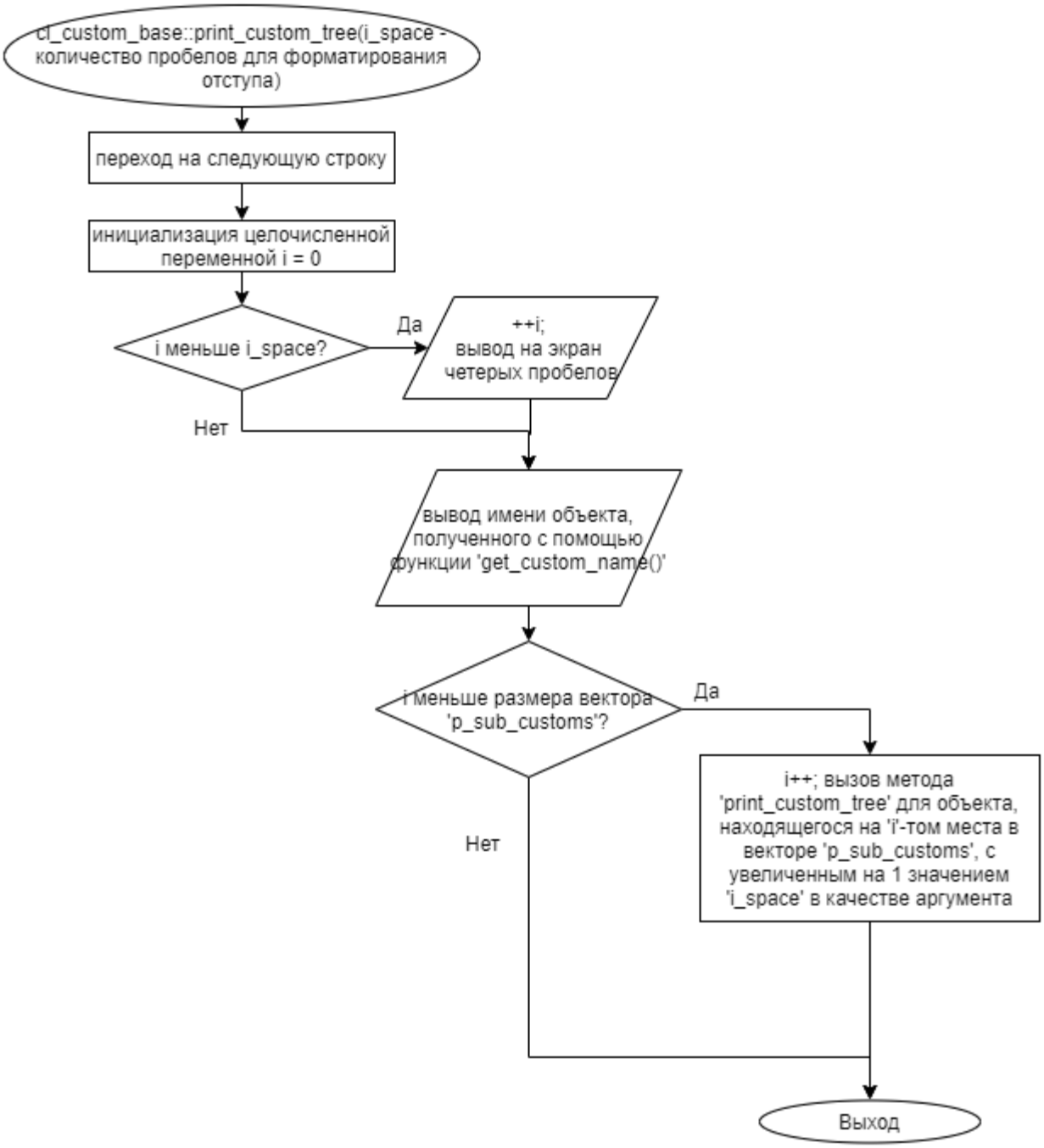

Рисунок 6 – Блок-схема алгоритма

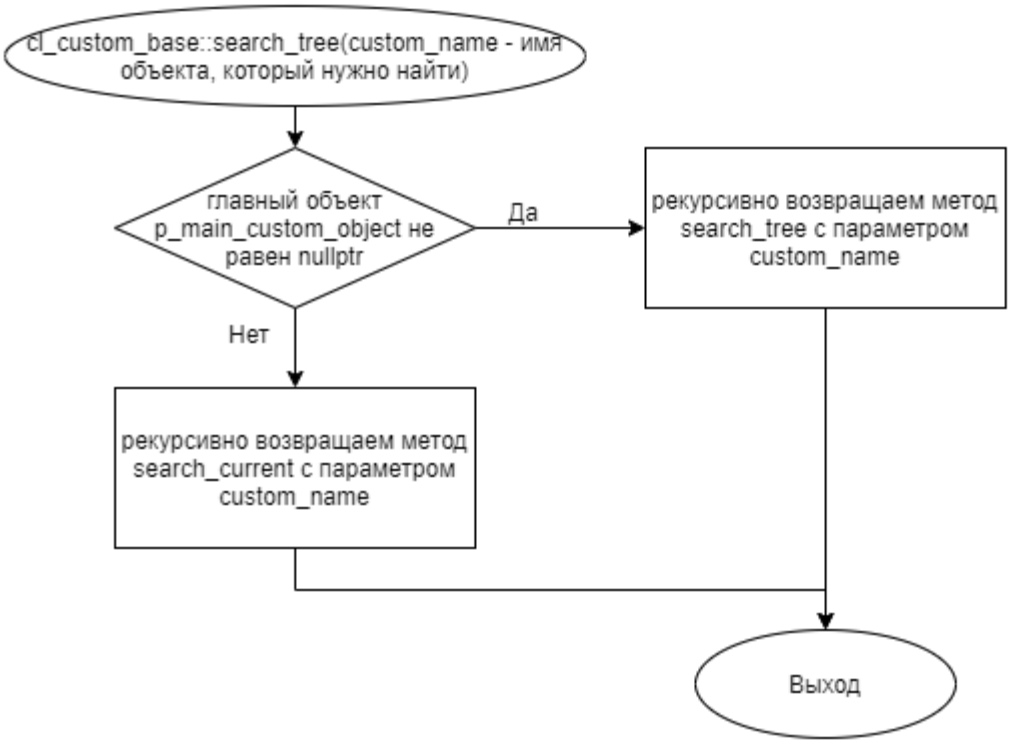

Рисунок 7 – Блок-схема алгоритма

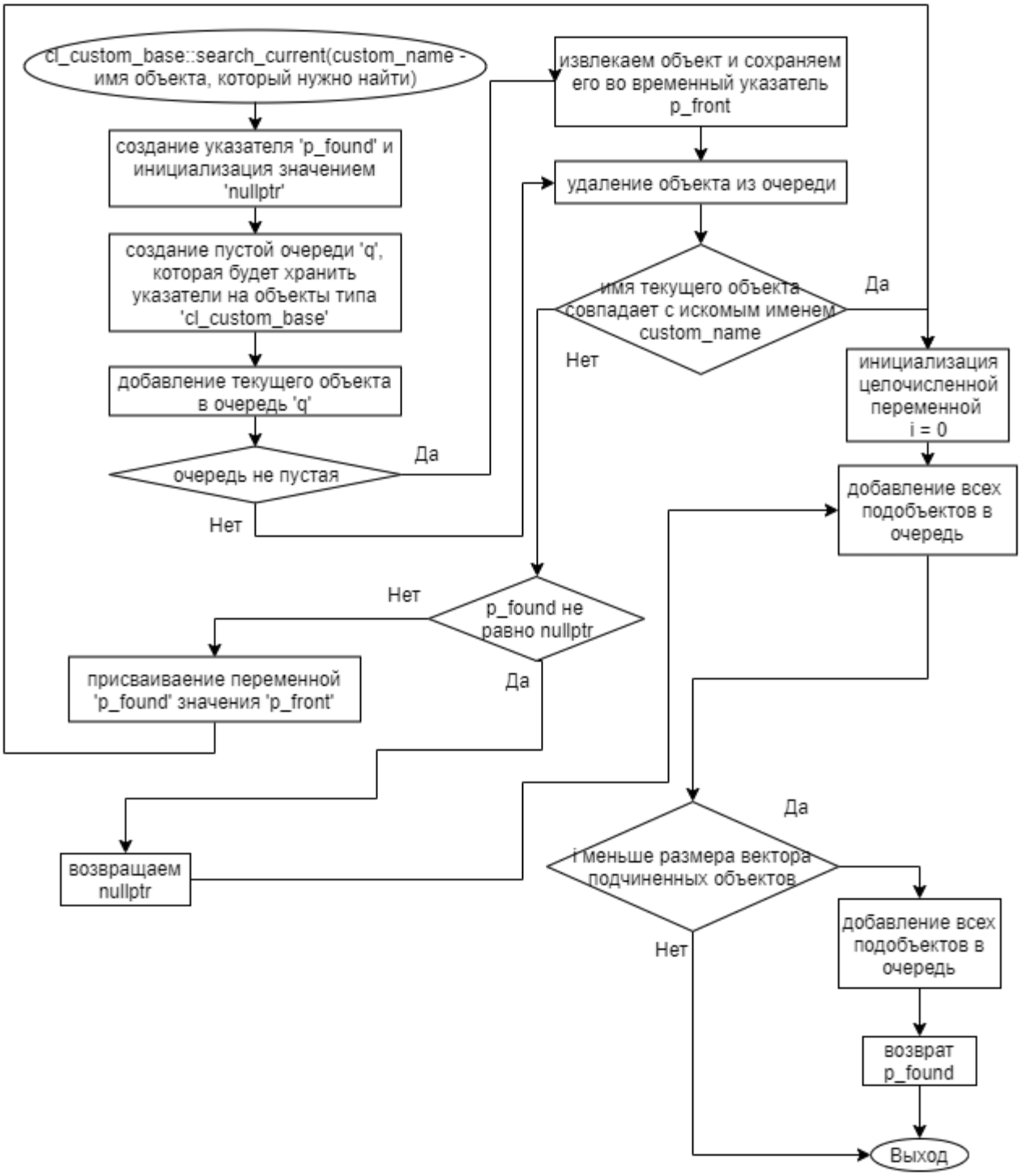

Рисунок 8 – Блок-схема алгоритма

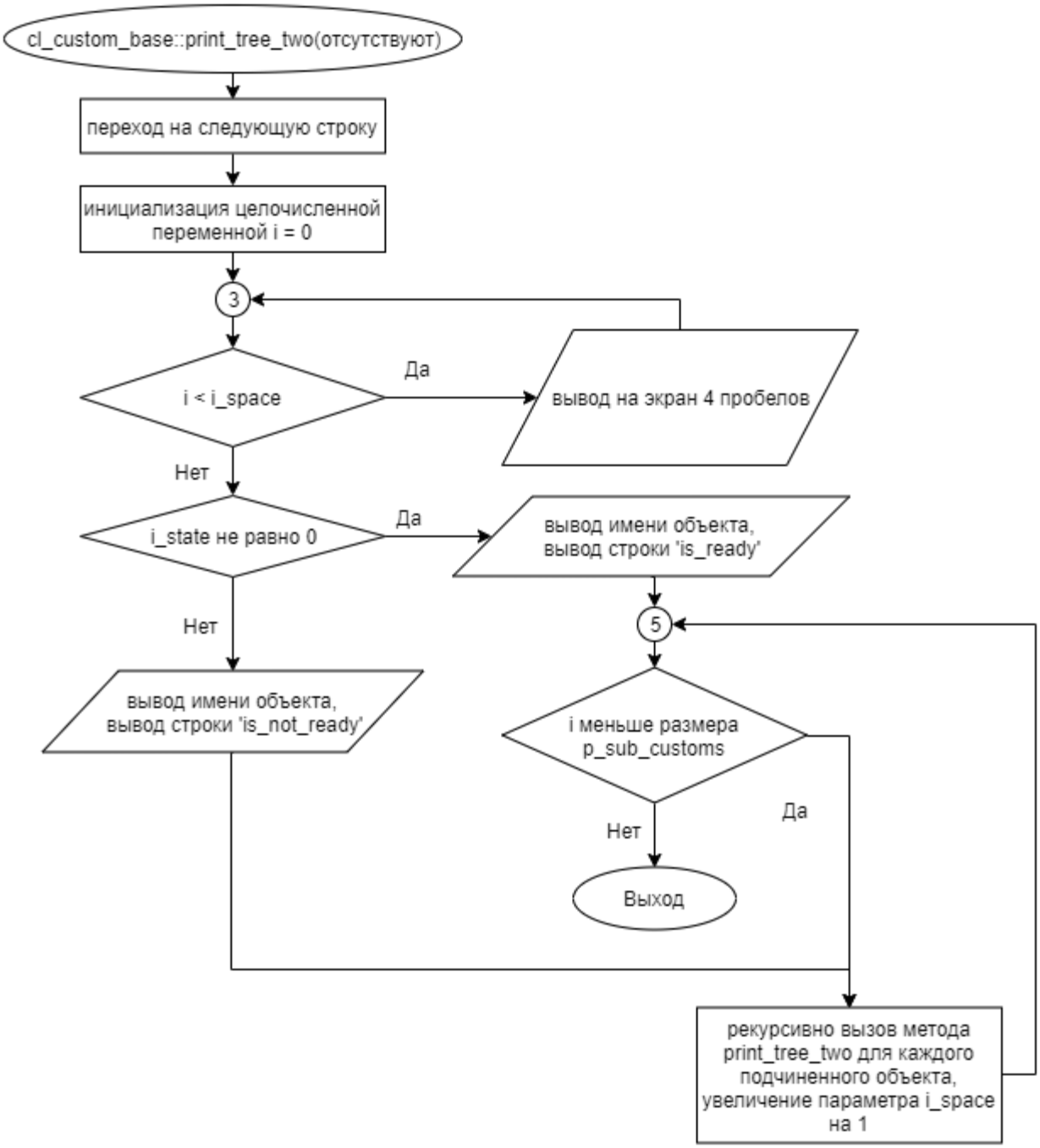

Рисунок 9 – Блок-схема алгоритма

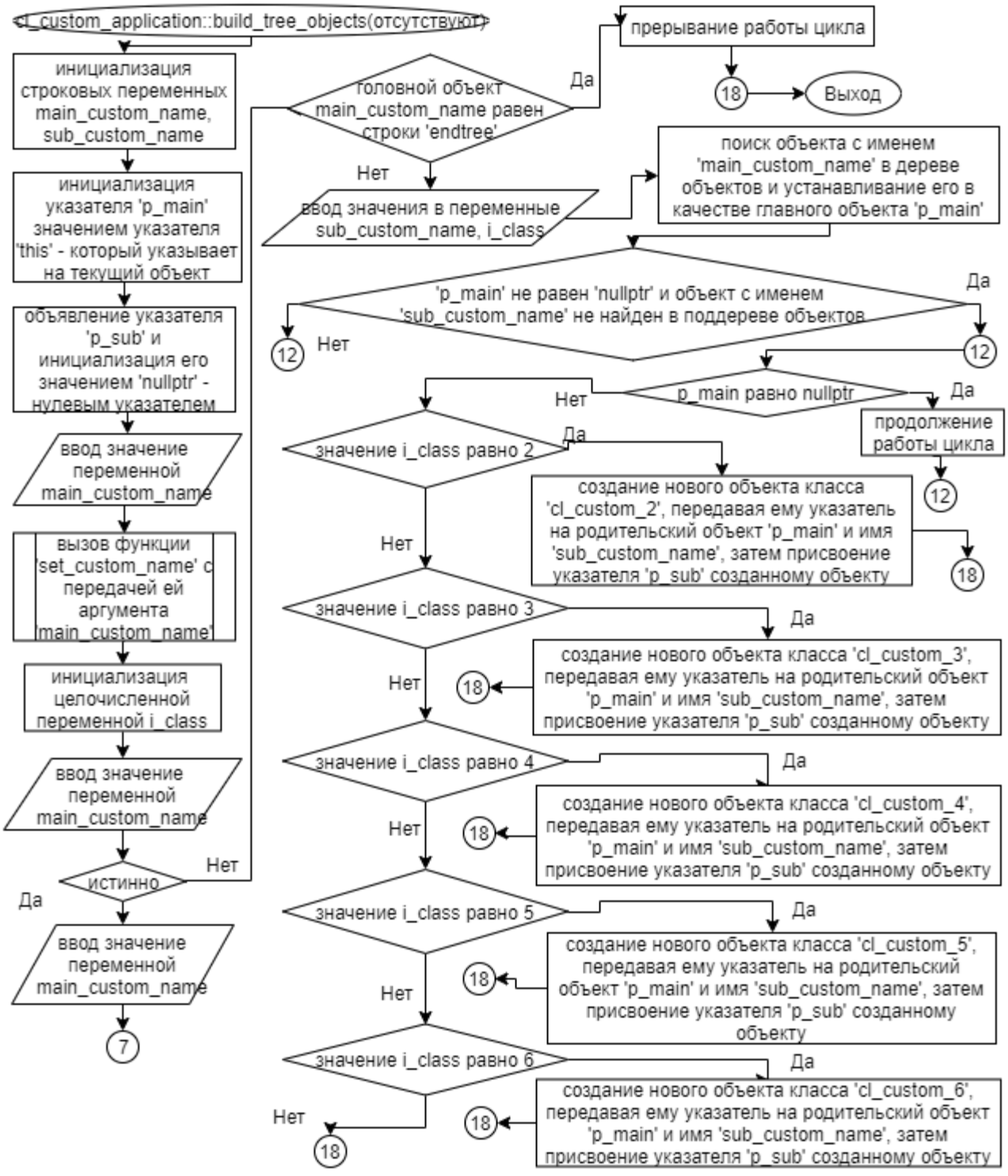

Рисунок 10 – Блок-схема алгоритма

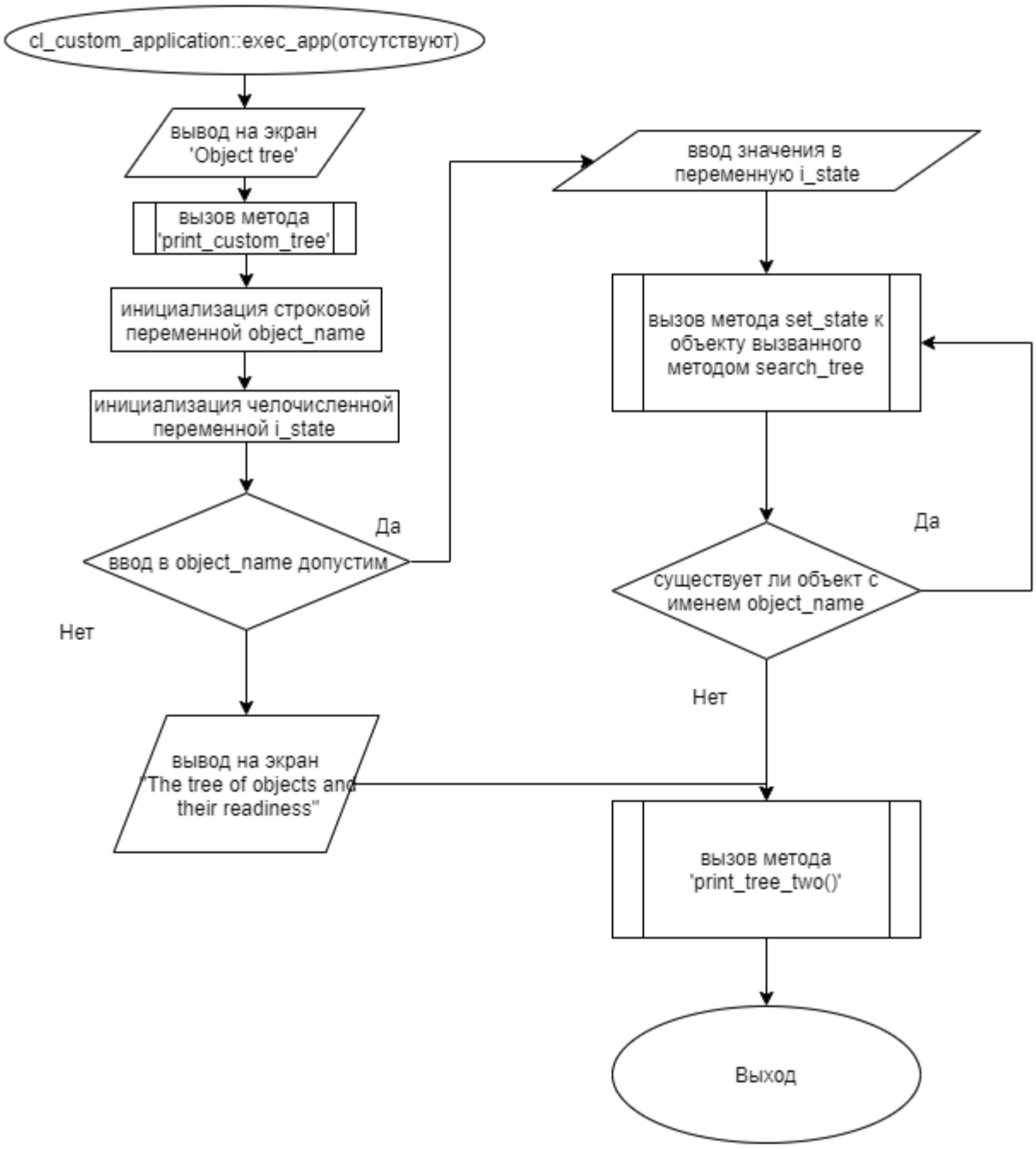

Рисунок 11 – Блок-схема алгоритма

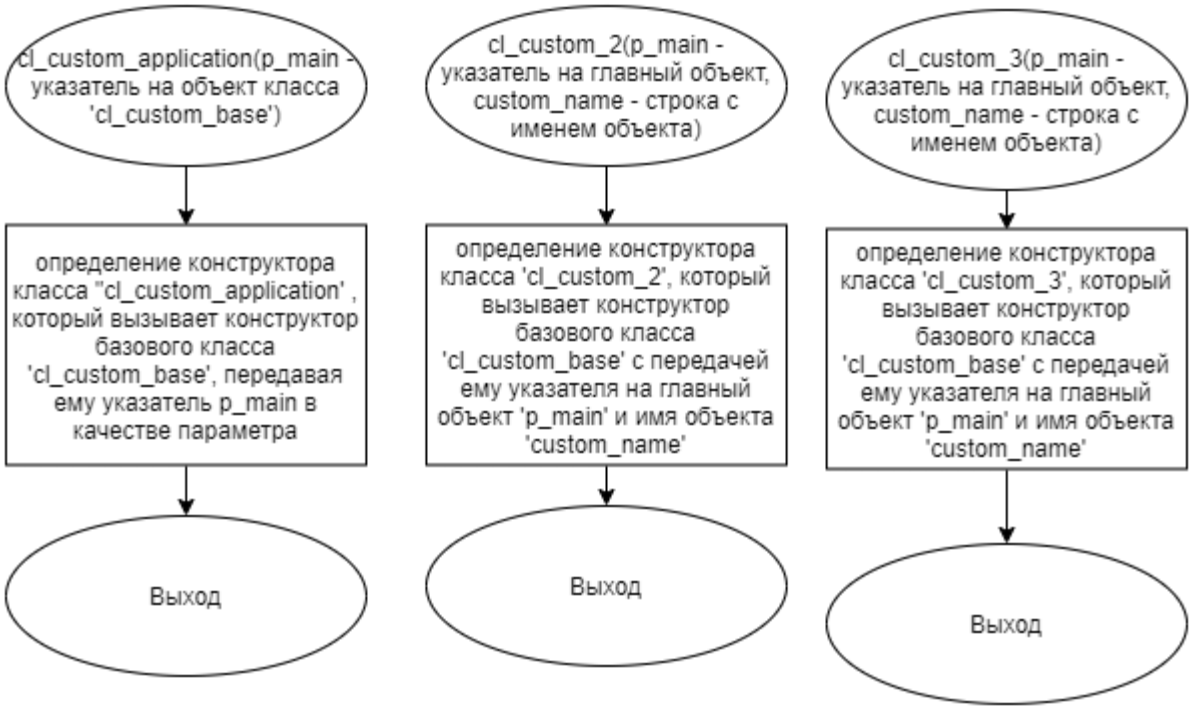

Рисунок 12 – Блок-схема алгоритма

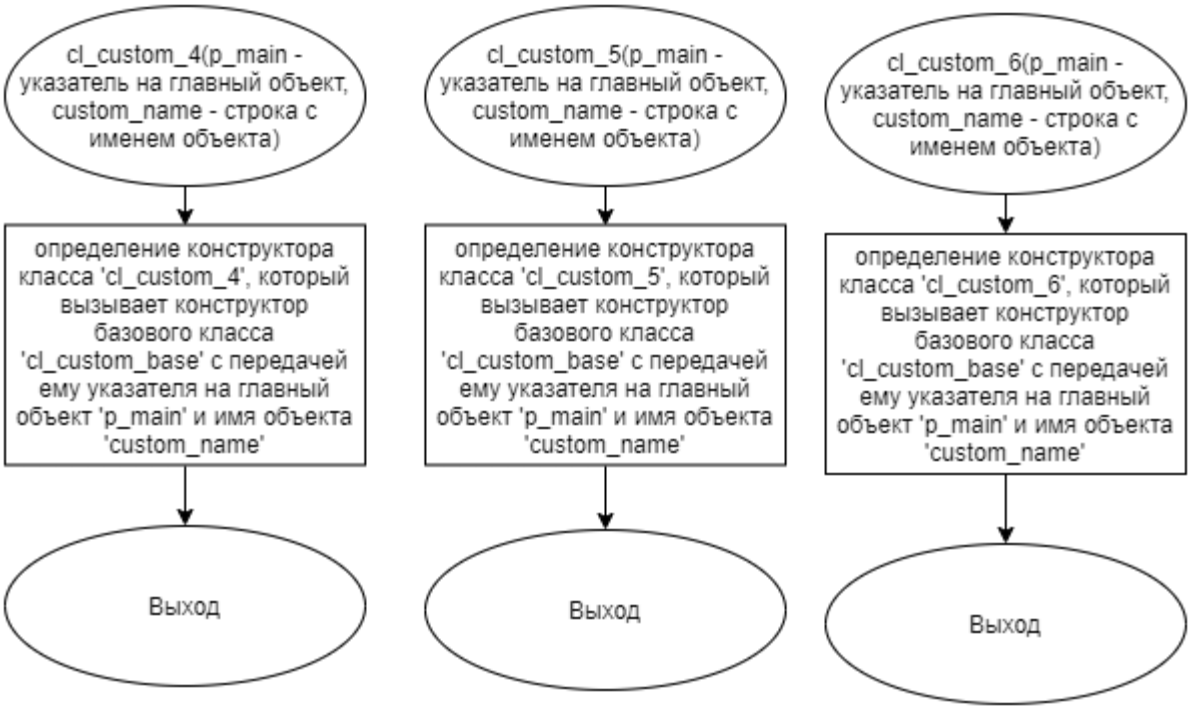

Рисунок 13 – Блок-схема алгоритма

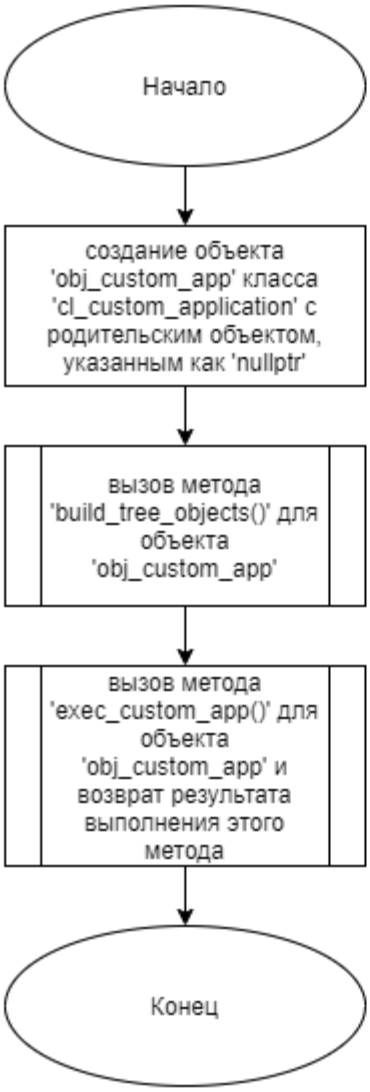

Рисунок 14 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_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   custom_name)   :
cl_custom_base(p_main, custom_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 {
public:
   cl_custom_2(cl_custom_base* p_main, string custom_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   custom_name)   :
cl_custom_base(p_main, custom_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 {
public:
   cl_custom_3(cl_custom_base* p_main, string custom_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   custom_name)   :
cl_custom_base(p_main, custom_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 {
public:
   cl_custom_4(cl_custom_base* p_main, string custom_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   custom_name)   :
cl_custom_base(p_main, custom_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 {
public:
   cl_custom_5(cl_custom_base* p_main, string custom_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   custom_name)   :
cl_custom_base(p_main, custom_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 {
public:
   cl_custom_6(cl_custom_base* p_main, string custom_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"

cl_custom_application::cl_custom_application(cl_custom_base*    p_main)    :
cl_custom_base(p_main) {}

void cl_custom_application::build_tree_objects() {
   string main_custom_name, sub_custom_name;

   cl_custom_base* p_main = this;

   cl_custom_base* p_sub = nullptr;

   cin >> main_custom_name;

   set_custom_name(main_custom_name);

   int i_class;

   while(true) {
      cin >> main_custom_name;
      if(main_custom_name == "endtree") {
         break;
      }
```

```
      cin >> sub_custom_name >> i_class;
      p_main = search_tree(main_custom_name);

      if(p_main   !=   nullptr   &&   p_main->search_tree(sub_custom_name)   ==
nullptr) {}

      switch(i_class) {
         case 2:
            p_sub = new cl_custom_2(p_main, sub_custom_name);
            break;
         case 3:
            p_sub = new cl_custom_3(p_main, sub_custom_name);
            break;
         case 4:
            p_sub = new cl_custom_4(p_main, sub_custom_name);
            break;
         case 5:
            p_sub = new cl_custom_5(p_main, sub_custom_name);
            break;
         case 6:
            p_sub = new cl_custom_6(p_main, sub_custom_name);
            break;
         default:
            break;
      }

   }
}

int cl_custom_application::exec_custom_app() {
   cout << "Object tree";
   print_custom_tree();
   string object_name;
   int i_state;
   while(cin >> object_name) {
      cin >> i_state;
      if(search_tree(object_name) != nullptr) {
         search_tree(object_name)->set_state(i_state);
      }
   }
   cout << endl << "The tree of objects and their readiness";
   print_tree_two();
   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 <iostream>
#include "cl_custom_base.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"

class cl_custom_application : public cl_custom_base {
public:
   cl_custom_application(cl_custom_base* p_main);
   void build_tree_objects();
   int exec_custom_app();
};
#endif
```

5.13 Файл cl_custom_base.cpp

Листинг 13 – cl_custom_base.cpp

```
#include "cl_custom_base.h"

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

   if(p_main_custom_object != nullptr) {
      p_main_custom_object->p_sub_customs.push_back(this);
   }
};

cl_custom_base::~cl_custom_base() {
   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 p_main_custom_object;
};

bool cl_custom_base::set_custom_name(string new_custom_name) {
   if(get_main_custom()     !=     nullptr     &&     p_main_custom_object-
>get_sub_custom(new_custom_name) != nullptr) {
      return false;
   }
   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 custom_name) {
   for(int i = 0; i < p_sub_customs.size(); i++) {
      if(p_sub_customs[i]->get_custom_name() == custom_name) {
         return p_sub_customs[i];
      }
   };

   return nullptr;
};

void cl_custom_base::print_custom_tree(int i_space) {
   cout << endl;
   for(int i = 0; i < i_space; ++i) {
      cout << "    ";
   }

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

}

void cl_custom_base::print_tree_two(int i_space) {
   cout << endl;
   for (int i = 0; i < i_space; ++i) {
      cout << "    ";
   }
   if (this->i_state != 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);
   }
}

cl_custom_base* cl_custom_base::search_current(string custom_name) {
   cl_custom_base* p_found = nullptr;
   queue<cl_custom_base*> q;

   q.push(this);

   while(!q.empty()) {
      cl_custom_base* p_front = q.front();
      q.pop();
```

```
      if(p_front->get_custom_name() == custom_name) {
         if(p_found != nullptr) {
            return nullptr;
         } else {
            p_found = p_front;
         }
      }

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

   return p_found;
}

cl_custom_base* cl_custom_base::search_tree(string custom_name) {
   if(p_main_custom_object != nullptr) {
      return p_main_custom_object->search_tree(custom_name);
   } else {
      return search_current(custom_name);
   }
}

void cl_custom_base::set_state(int i_state) {
   if(p_main_custom_object == nullptr || p_main_custom_object->i_state != 0)
{
      this->i_state = i_state;
   }

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

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>

using namespace std;
```

```
class cl_custom_base {
private:
   string custom_name;
   cl_custom_base* p_main_custom_object;
   vector<cl_custom_base*> p_sub_customs;
   int i_state = 0;

public:
   cl_custom_base(cl_custom_base*   p_main,   string   custom_name   =   "Base
Object");
   ~cl_custom_base();
   bool set_custom_name(string new_custom_name);
   string get_custom_name();
   cl_custom_base* get_main_custom();
   cl_custom_base* get_sub_custom(string custom_name);
   cl_custom_base* search_current(string);
   cl_custom_base* search_tree(string);
   void set_state(int);
   void print_custom_tree(int numb = 0);
   void print_tree_two(int numb = 0);
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <string>
#include <iostream>
#include "cl_custom_application.h"

int main() {
   cl_custom_application obj_custom_app(nullptr);
   obj_custom_app.build_tree_objects();
   return(obj_custom_app.exec_custom_app());
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 22.

Таблица 22 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
app_root app_root object_01 3 app_root object_02 2 object_02 object_04 3 object_02 object_05 5 object_01 object_07 2 endtree app_root 1 object_07 3 object_01 1 object_02 -2 object_04 1	Object tree app_root object_01 object_07 object_02 object_04 object_05 The tree of objects and their readiness app_root is ready object_01 is ready object_07 is not ready object_02 is ready object_04 is ready object_05 is not ready	Object tree app_root object_01 object_07 object_02 object_04 object_05 The tree of objects and their readiness app_root is ready object_01 is ready object_07 is not ready object_02 is ready object_04 is ready object_05 is not ready

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).