Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Для организации иерархического построения объектов необходимо разработать базовый класс, который содержит функционал и свойства для построения иерархии объектов. В последующем, в приложениях использовать этот класс как базовый для всех создаваемых классов. Это позволит включать любой объект в состав дерева иерархии объектов.

Каждый объект на дереве иерархии имеет свое место и наименование. Не допускается для одного головного объекта одинаковые наименования в составе подчиненных объектов.

Создать базовый класс со следующими элементами:

- свойства:
  - наименование объекта (строкового типа);
  - указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно nullptr);
  - динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.
- функционал:
  - параметризированный конструктор с параметрами: указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии; строкового типа, содержащий наименование создаваемого объекта (имеет значение по умолчанию);
  - метод редактирования имени объекта. Один параметр строкового типа, содержит новое наименование объекта. Если нет дубляжа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь»;
  - метод получения имени объекта;

- метод получения указателя на головной объект текущего объекта;
- метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- метод получения указателя на непосредственно подчиненный объект по его имени. Если объект не найден, то возвращает nullptr. Один параметр строкового типа, содержит наименование искомого подчиненного объекта.

Для построения дерева иерархии объектов в качестве корневого объекта используется объект приложение. Класс объекта приложения наследуется от базового класса. Объект приложение реализует следующий функционал:

- метод построения исходного дерева иерархии объектов (конструирования моделируемой системы);
- метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Написать программу, которая последовательно строит дерево иерархии объектов, слева направо и сверху вниз. Переход на новый уровень происходит только от правого (последнего) объекта предыдущего уровня. Для построения дерева использовать объекты двух производных классов, наследуемых от базового. Исключить создание объекта если его наименование совпадает с именем уже имеющегося подчиненного объекта у предполагаемого головного. Исключить добавление нового объекта, не последнему подчиненному предыдущего уровня.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

```
int main ( )
{
      cl_application   ob_cl_application ( nullptr ); // создание корневого
объекта
       ob_cl_application.build_tree_objects ( );          // конструирование
```

```
системы, построение дерева объектов
    return ob_cl_application.exec_app ( );         // запуск системы
}
```

Наименование класса cl_application и идентификатора корневого объекта ob_cl_application могут быть изменены разработчиком.

Все версии курсовой работы имеют такую основную функцию.

1.1 Описание входных данных

Первая строка:

«имя корневого объекта»

Вторая строка и последующие строки:

«имя головного объекта» «имя подчиненного объекта»

Создается подчиненный объект и добавляется в иерархическое дерево. Если «имя головного объекта» равняется «имени подчиненного объекта», то новый объект не создается и построение дерева объектов завершается.

Пример ввода:

```
Object_root
Object_root Object_1
Object_root Object_2
Object_root Object_3
Object_3 Object_4
Object_3 Object_5
Object_6 Object_6
```

Дерево объектов, которое будет построено по данному примеру:

```
Object_root
    Object_1
    Object_2
    Object_3
        Object_4
        Object_5
```

1.2 Описание выходных данных

Первая строка:

«имя корневого объекта»

Вторая строка и последующие строки имена головного и подчиненных объектов очередного уровня разделенных двумя пробелами.

«имя головного объекта» «имя подчиненного объекта»[[«имя подчиненного объекта»] …….]

Пример вывода:

```
Object_root
Object_root  Object_1  Object_2  Object_3
Object_3  Object_4  Object_5
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект ob_cl_application класса Cl_application предназначен для конструирования и запуска системы;
- Объект стандартного потока ввода с клавиатуры cin;
- Объект стандартного потока вывода на экран cout;
- Условный опреатор if..else;
- Оператор цикла for;
- Оператор цикла с предусловием while.

Класс Cl_base:

- свойства/поля:
    - поле Наименование объекта:
        - наименование — s_object_name;
        - тип — string;
        - модификатор доступа — private;
    - поле Указатель на головной объект для текущего объекта:
        - наименование — p_head_object;
        - тип — Cl_base*;
        - модификатор доступа — private;
    - поле Динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии:
        - наименование — subordinate_objects;
        - тип — vector <Cl_base*>;
        - модификатор доступа — private;
- функционал:
    - метод Cl_base — Параметризированый конструктор;

- метод change_object_name — Метод редактирования имени объекта;
- метод get_object_name — Метод получения имени объекта;
- метод get_head_object — Метод получения указателя на головной объект текущего объекта;
- метод show_object_tree — Метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- метод get_sub_object_by_name — Метод получения указателя на непосредственно подчиненный объект по его имени.

Класс Cl_child:

- функционал:
  - метод Cl_child — Параметризованный конструктор.

Класс Cl_application:

- функционал:
  - метод Cl_application — Параметризированный конструктор;
  - метод build_tree_objects — Метод построения исходного дерева иерархии объектов;
  - метод exec_app — Метод запуска приложения.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	Cl_base			Базовый класс	
		Cl_child	public		2
		Cl_application	public		3
2	Cl_child			Дочерний класс	
3	Cl_application			Класс приложение	

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_base

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на родительский объект класса Cl_base. Строка s_object_name обозначающая наименование объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса Cl_base

№	Предикат	Действия	№ перехода
1		Закрытому свойству p_head_object присваивается значение параметра p_head_object	2
2		Закрытому свойству s_object_name присваивается значение параметра s_object_name	3
3	p_head_object существует?	Добавление текущего объекта в массив дочерних объектов (subordinate_objects) объекта по адресу p_head_object	∅
			∅

3.2 Алгоритм конструктора класса Cl_child

Функционал: Параметризованный конструктор.

Параметры: Указатель p_head_object на родительский объект класса Cl_base. Строка s_object_name обозначающая наименование объекта.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса Cl_child

№	Предикат	Действия	№ перехода
1		Передача параметров p_head_object и s_object_name параметризированному конструктору класса Cl_base	∅

3.3 Алгоритм конструктора класса Cl_application

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на родительский объект класса Cl_base. Строка s_object_name обозначающая наименование объекта.

Алгоритм конструктора представлен в таблице 4.

Таблица 4 – Алгоритм конструктора класса Cl_application

№	Предикат	Действия	№ перехода
1		Передача параметров p_head_object и s_object_name параметризированному конструктору класса Cl_base	2
2		Ввод с клавиатуры значения параметра s_object_name	3
3		Вызов метода change_object_name с параметром s_object_name	∅

3.4 Алгоритм метода build_tree_objects класса Cl_application

Функционал: Метод построения исходного дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода build_tree_objects класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление указателя last_object на объект класса Cl_base и инициализация текущим объектом	2

№	Предикат	Действия	№ перехода
2		Объявление переменных sub и head строкового типа	3
3		Ввод с клавиатуры значений head и sub	4
4	head не равно sub?		5
			∅
5	Результат метода get_object_name объекта по адресу last_object равен head или (результат метода get_head_object объекта по адресу last_object не равен nullptr и результат метода get_object_name по адресу результата метода get_head_object по адресу объекта last_object равен head)?	Объявление указателя head_object	6
			9
6	Результат метода get_object_name объекта по адресу last_object равен head?	Указателю head_object присваивается адрес last_object	7
		Указателю head_object присваивается результат метода get_head_object объекта по адресу last_object	7
7	Результат метода get_sub_object_by_name с параметром sub объекта по адресу head_object равен	Объявление указателя subordinate_object и инициализация новым объектом класса Cl_child с параметрами head_object и sub	8

№	Предикат	Действия	№ перехода
	nullptr?		
			9
8		Указателю last_object присваивается адрес subordinate_object	9
9		Ввод с клавиатуры значений head и sub	4

3.5 Алгоритм метода exec_app класса Cl_application

Функционал: Метод запуска приложения.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода exec_app класса Cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода get_object_name и вывод результата на экран	2
2		Вызов метода show_object_tree	3
3		Возврат 0	∅

3.6 Алгоритм метода change_object_name класса Cl_base

Функционал: Метод редактирования имени объекта.

Параметры: Строка s_object_name с новым наименованием объекта.

Возвращаемое значение: Булевский тип обозначающий возможность смены имени объекта.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода change_object_name класса Cl_base

№	Предикат	Действия	№ перехода
1	s_object_name пустая?	Вернуть false	∅
		Объявление указателя subordinate_object на объект класса Cl_base	2
2	subordinate_object считывается из subordinate_objects?		3
			4
3	Результат метода get_object_name() объекта subordinate_object равен s_object_name?	Вернуть false	∅
		Считать следующий subordinate_object из subordinate_objects	2
4		Методу s_object_name присвоить значение параметра s_object_name	5
5		Вернуть true	∅

3.7 Алгоритм метода get_object_name класса Cl_base

Функционал: Метод получения имени объекта.

Параметры: нет.

Возвращаемое значение: Строка наименования объекта.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода get_object_name класса Cl_base

№	Предикат	Действия	№ перехода
1		Вернуть значение s_object_name	∅

3.8 Алгоритм метода get_head_object класса Cl_base

Функционал: Метод получения указателя на головной объект текущего объекта.

Параметры: нет.

Возвращаемое значение: Указатель на головной объект класса Cl_base.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_head_object класса Cl_base

№	Предикат	Действия	№ перехода
1		Вернуть значение p_head_object	∅

3.9 Алгоритм метода show_object_tree класса Cl_base

Функционал: Метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз.

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода show_object_tree класса Cl_base

№	Предикат	Действия	№ перехода
1	Размер массива subordinate_objects > 0?	Вывод переоса строки и результата метода get_object_name	2
			∅
2		Объявление указателя subordinate_object на объект класса Cl_base	3
3	subordinate_object считывается из subordinate_objects?	Вывод двух пробелов и результата метода get_object_name объекта по адресу subordinate_object	4

№	Предикат	Действия	№ перехода
		Объявление указателя subordinate_object на объект класса Cl_base	5
4		Считать следующий subordinate_object из subordinate_objects	3
5	subordinate_object считывается из subordinate_objects?		6
			∅
6	Размер массива subordinate_objects объекта по адресу subordinate_object > 0?	Вызов метода show_object_tree объекта по адресу subordinate_object	7
			7
7		Считать следующий subordinate_object из subordinate_objects	5

3.10 Алгоритм метода get_sub_object_by_name класса Cl_base

Функционал: Метод получения указателя на непосредственно подчиненный объект по его имени.

Параметры: Строка s_object_name с наименованием объекта.

Возвращаемое значение: Указатель на объект класса Cl_base.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода get_sub_object_by_name класса Cl_base

№	Предикат	Действия	№ перехода
1	s_object_name не пустая?	Объявление указателя subordinate_object на объект класса Cl_base	2
			4

№	Предикат	Действия	№ перехода
2	subordinate_object считывается из subordinate_objects?		3
			4
3	Результат метода get_object_name() объекта subordinate_object равен s_object_name?	Вернуть subordinate_object	∅
		Считать следующий subordinate_object из subordinate_objects	2
4		Вернуть nullptr	∅

3.11 Алгоритм функции main

Функционал: Построение дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм функции представлен в таблице 12.

Таблица 12 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Объявление объекта ob_cl_application класса Cl_application с параметром nullptr	2
2		Вызов метода build_tree_objects объекта ob_cl_application	3
3		Возврат значения метода exec_app объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-6.

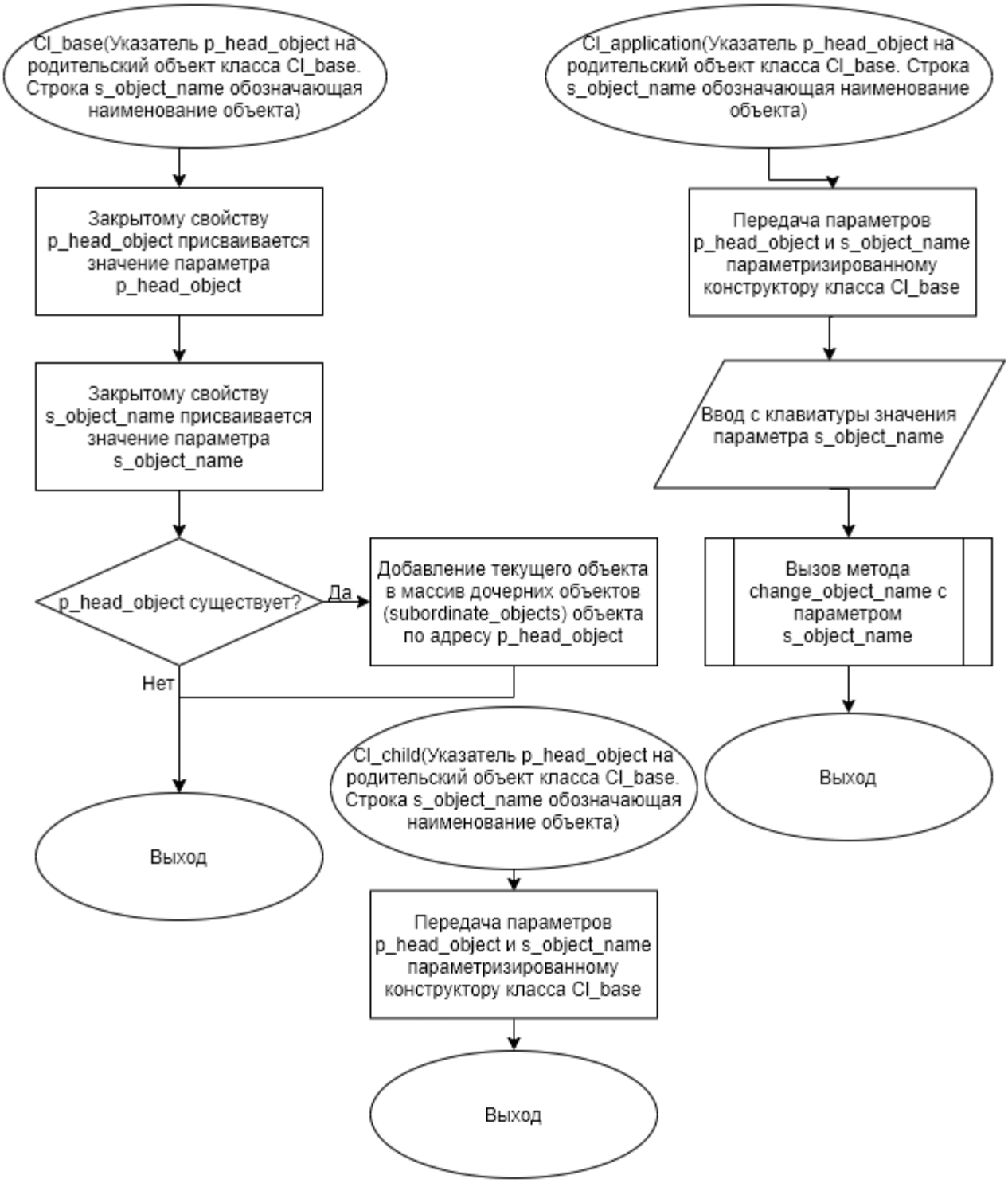

Рисунок 1 – Блок-схема алгоритма

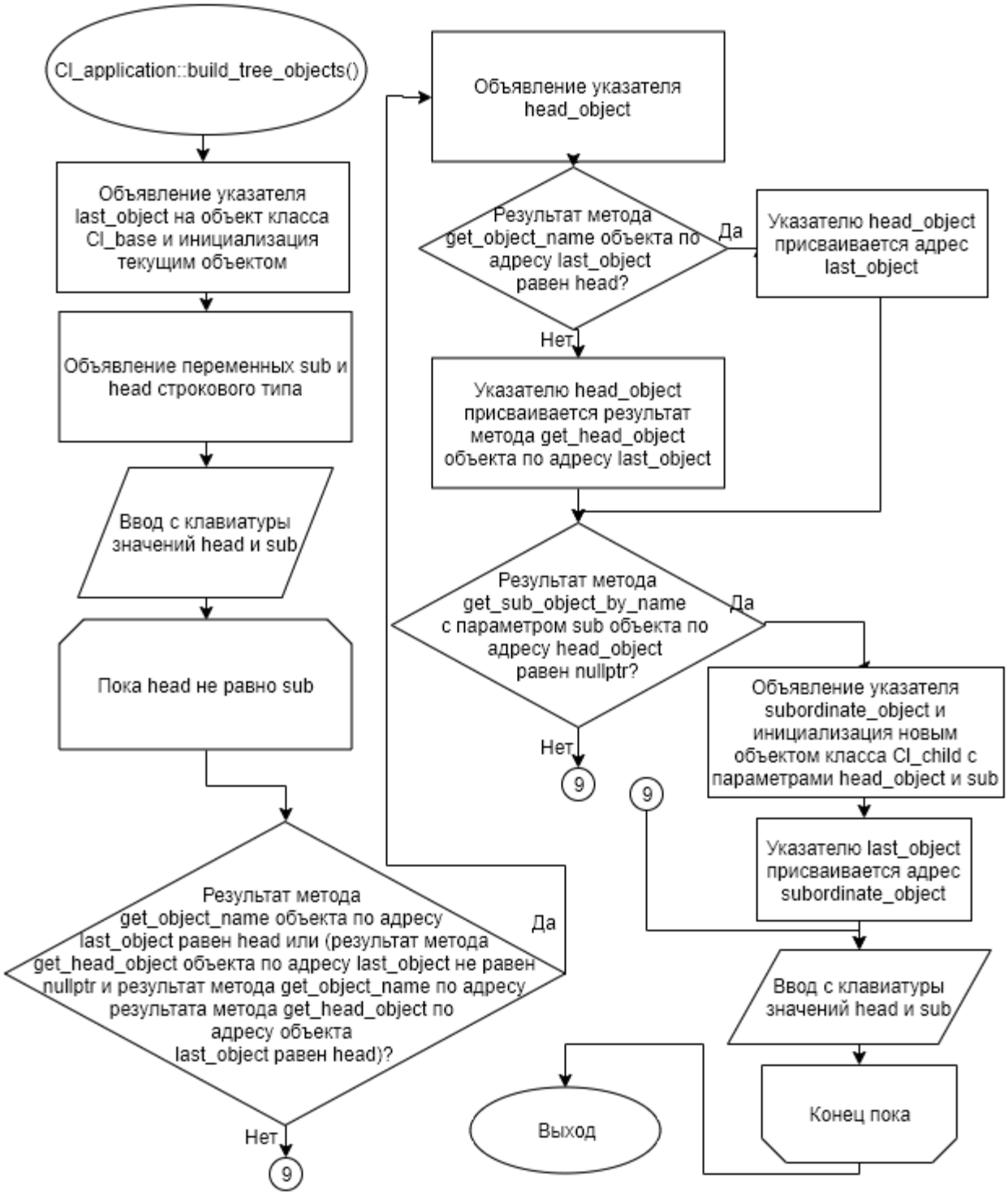

Рисунок 2 – Блок-схема алгоритма

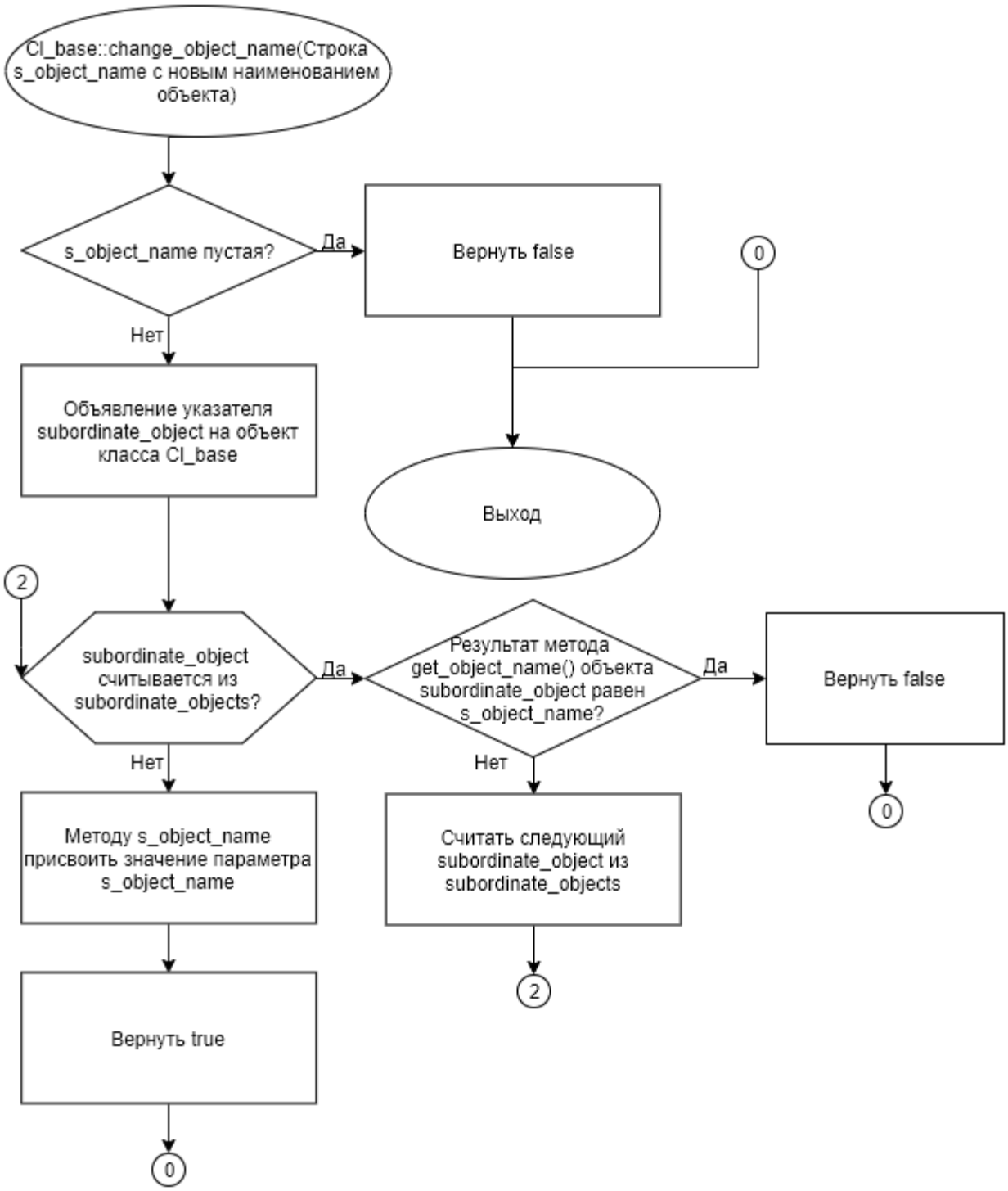

Рисунок 3 – Блок-схема алгоритма

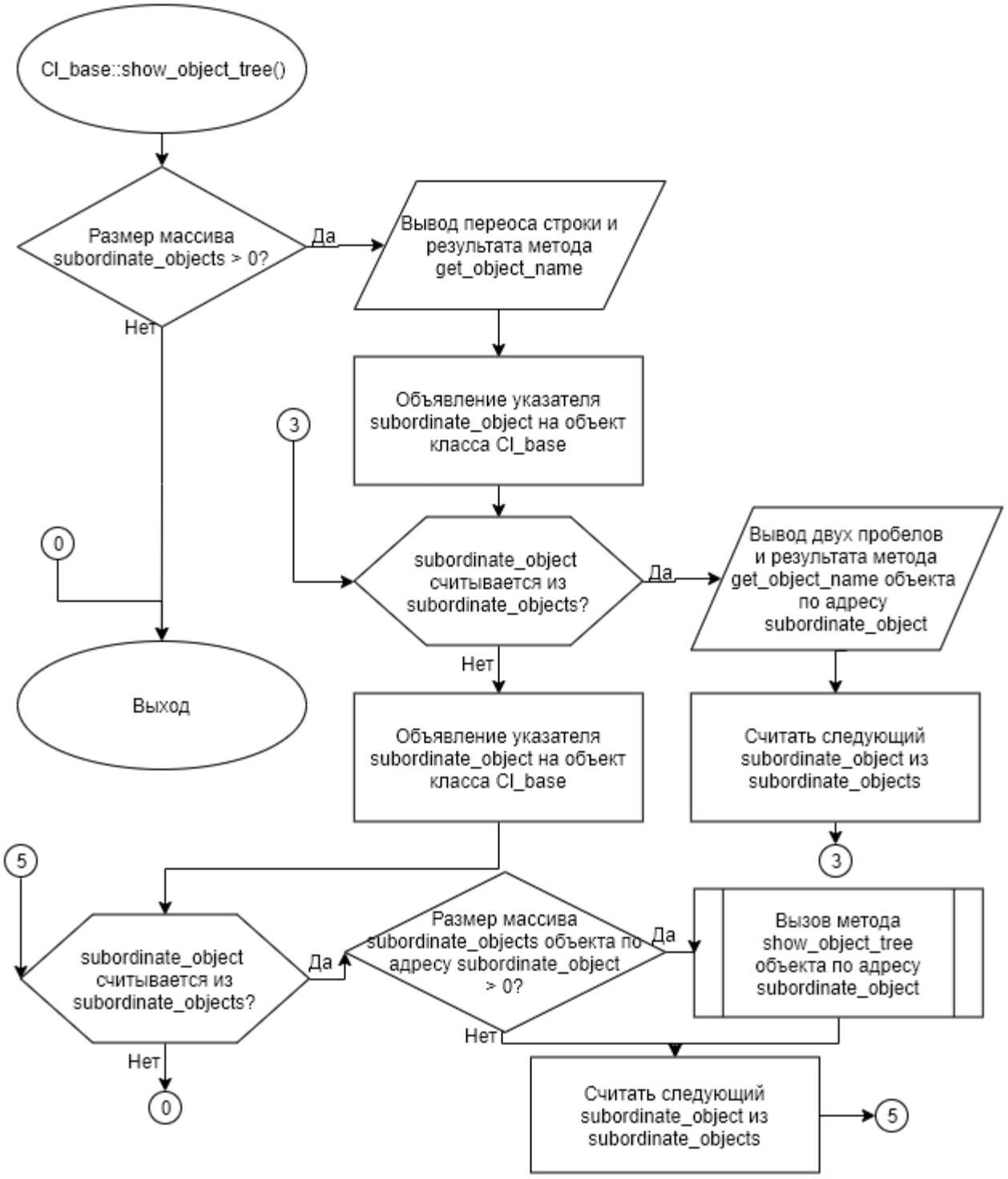

Рисунок 4 – Блок-схема алгоритма

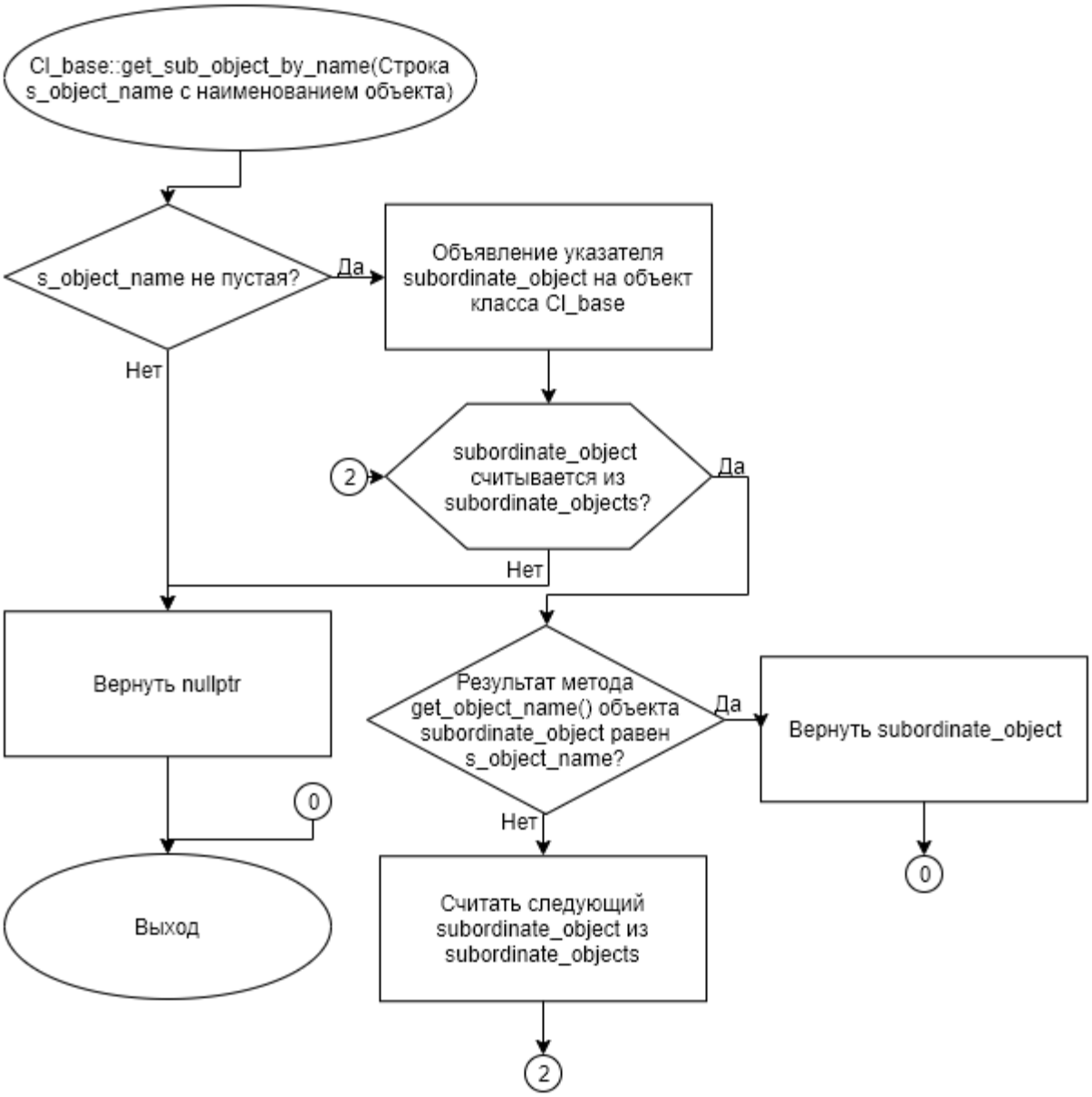

Рисунок 5 – Блок-схема алгоритма

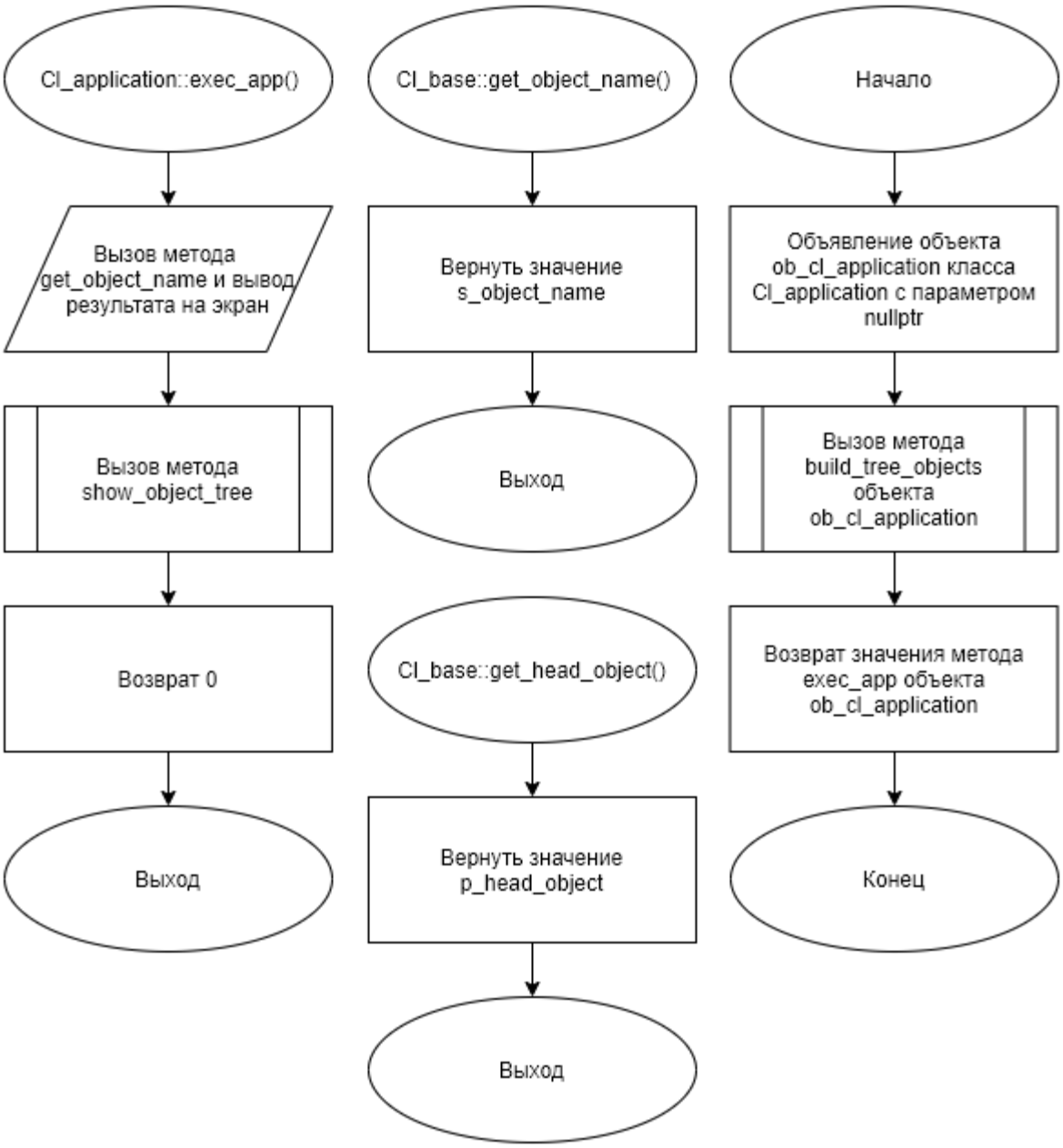

Рисунок 6 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл Cl_application.cpp

Листинг 1 – Cl_application.cpp

```
#include "Cl_application.h"

Cl_application::Cl_application(Cl_base*         p_head_object,         string
s_name_object):Cl_base(p_head_object, s_name_object)
{
   cin >> s_name_object;//ввод имени объекта
   change_object_name(s_name_object);//изменение имени объекта
}

void Cl_application::build_tree_objects()
{
   Cl_base* last_object = this;//Создание указателя на текущий объект
   string sub, head;//Создание строк
   cin >> head >> sub;//Ввод значения строк
   while (head != sub) {//пока значения строк не равны
      if  (last_object  ->  get_object_name()  ==  head  ||  (last_object  ->
get_head_object() != nullptr &&
         last_object  ->  get_head_object()  ->  get_object_name()  ==  head))
{//если имя равно head или есть родительский объект и имя родительсякого
равно head
         auto  *head_object  =  last_object  ->  get_object_name()  ==  head
?//новому указателю присваиваем значение текущего, если имя равно строке
head
            last_object : last_object -> get_head_object();
         if (head_object -> get_sub_object_by_name(sub) == nullptr) {//если
имя не найдено, то
            auto    *subordinate_object    =    new    Cl_child(head_object,
sub);//Создаем новый объект
            last_object   =   subordinate_object;//новый   объект   становится
последним
         }
      }
      cin >> head >> sub;//Ввод значения строк
   }

}

int Cl_application::exec_app()
{
```

```
   cout << get_object_name();//вывод имени головного объекта
   show_object_tree();//вывод дерева объектов
   return 0;
}
```

5.2 Файл Cl_application.h

Листинг 2 – Cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "Cl_child.h"

class Cl_application: public Cl_base//наследование класса
{
public:
   Cl_application(Cl_base*          p_head_object,          string          =
"Base_object");//параметризированный конструктор
   void build_tree_objects();//метод построения дерева
   int exec_app();//метод запуска системы
};

#endif
```

5.3 Файл Cl_base.cpp

Листинг 3 – Cl_base.cpp

```
#include "Cl_base.h"

Cl_base::Cl_base(Cl_base* p_head_object, string s_object_name)
{
   this   ->   p_head_object   =   p_head_object;//присвоение   указателя   на
родительский объект
   this -> s_object_name = s_object_name;//присвоение имени объекта
   if ( p_head_object ) {//есть родительский объект?
      p_head_object   ->   subordinate_objects.push_back(this);//добавить   в
производные объекты
   }
}

bool Cl_base::change_object_name(string s_object_name)
{
   if (s_object_name.empty()) {//пустая строка?
      return false;
```

```
   }
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
      if (subordinate_object -> get_object_name() == s_object_name) {//если
имя равно искомому
         return false;
      }
   }
   this -> s_object_name = s_object_name;//сменить имя
   return true;
}

string Cl_base::get_object_name()
{
   return s_object_name;//вернуть имя объекта
}

Cl_base* Cl_base::get_head_object()
{
   return p_head_object;//вернуть указатель на родительский объект
}

void Cl_base::show_object_tree()
{
   if (subordinate_objects.size() > 0){//есть подчиненные объекты?
      cout << endl << get_object_name();//вывод имени объекта
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
обектов из списка
         cout << "   " << subordinate_object -> get_object_name();//вывод
имени обьета
      }
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
         if (subordinate_object -> subordinate_objects.size() > 0) {//есть
подчиненные объекты?
            subordinate_object -> show_object_tree();//уход в рекурсию
         }
      }
   }
}

Cl_base* Cl_base::get_sub_object_by_name(string s_object_name)
{
   if (!s_object_name.empty()) {//строка не пустая?
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
         if (subordinate_object -> get_object_name() == s_object_name) {//имя
равно искомому?
            return subordinate_object;//вернуть указатель на подчиненный
объект
         }
      }
   }
   return nullptr;
}
```

5.4 Файл Cl_base.h

Листинг 4 – Cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Cl_base//наименование класса
{
public:
   Cl_base     (Cl_base*     p_head_object,     string     s_object_name     =
"Base_object");//параметризированный конструктор
   bool change_object_name(string);//метод изменения имени
   string get_object_name();//метод получения имени
   Cl_base*  get_head_object();//метод  получения  указателя  на  родительский
объект
   void show_object_tree();//метод вывода дерева объектов
   Cl_base* get_sub_object_by_name(string);//метод поиска объекта по имени

private:
   string s_object_name;//имя объекта
   Cl_base* p_head_object;//указатель на родительский объект
   vector <Cl_base*> subordinate_objects;//подчиненные объекты
};



#endif
```

5.5 Файл Cl_child.cpp

Листинг 5 – Cl_child.cpp

```
#include "Cl_child.h"

Cl_child::Cl_child(Cl_base*               p_head_object,               string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.6 Файл Cl_child.h

Листинг 6 – Cl_child.h

```
#ifndef __CL_CHILD__H
#define __CL_CHILD__H
#include "Cl_base.h"

class Cl_child : public Cl_base//наследование класса
{
public:
   Cl_child(Cl_base*, string);//параметризированный конструктор
};

#endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "Cl_application.h"

int main()
{
   Cl_application ob_cl_application(nullptr);//создание объекта приложения
   ob_cl_application.build_tree_objects();//конструирование системы
   return ob_cl_application.exec_app();//запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 13.

Таблица 13 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_3 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5
Object_root Object_root Object_1 Object_1 Object_12 Object_1 Object_36 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_3 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_1 Object_12 Object_36	Object_root Object_root Object_1 Object_1 Object_12 Object_36
Object_root Object_root Object_1 Object_5 Object_5 Object_root Object_2	Object_root Object_root Object_1	Object_root Object_root Object_1

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).