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 cout класса потокового вывода предназначен для функционирования системы;
- объект cin класса потокового ввода предназначен для функционирования системы;
- объект obj_custom_app класса cl_custom_application предназначен для для запуска приложения;
- объект p_sub_customs класса cl_custom_1 предназначен для объектов элемента дерева класса cl_custom_1;
- 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 вектор указателей на подпользовательские объекты:
        - наименование — p_sub_customs;

        - тип — vector<cl_custom_base*>;
        - модификатор доступа — private;
- функционал:
    - 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 по имени;
    - метод print_custom_tree — вывод информации о пользовательских объектах и их иерархии на экран.

Класс cl_custom_1:

- функционал:
    - метод cl_custom_1 — конструктор.

Класс cl_custom_application:

- функционал:
    - метод cl_custom_application — конструктор;
    - метод build_tree_objects — построение иерархии пользовательских объектов внутри приложения;
    - метод exec_custom_app — выполнение пользовательского приложения.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_custom_base			основная логика управление пользовательскими объектами и их иерархией	
		cl_custom_1	public		2
		cl_custom_application	public		3
2	cl_custom_1			наследуется от 'cl_custom_base', представляет определенный тип пользовательского объекта и используется для создания экземпляров этого типа	
3	cl_custom_application			наследуется от 'cl_custom_base', представляет пользовательское приложение и содержит логику его выполнения	

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: инициализация нового пользовательского объекта с указанным именем и родительским объектом.

Параметры: cl_custom_base* p_main - указатель на родительский объект, string custom_name - имя пользователя.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_custom_base

№	Предикат	Действия	№ перехода
1		присваивание значения 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'	∅

№	Предикат	Действия	№ перехода
			∅

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: возврат имени текущего пользовательского объекта.

Параметры: отсутствуют.

Возвращаемое значение: string custom_name - имя текущего пользовательского объекта.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_custom_name класса cl_custom_base

№	Предикат	Действия	№ перехода
1		возврат имени текущего пользовательского объекта - string custom_name	∅

3.4 Алгоритм метода set_custom_name класса cl_custom_base

Функционал: установка нового имени для текущего объекта, предотвращая дублирование имен в иерархии объектов.

Параметры: string new_custom_name - новое имя пользовательского объекта.

Возвращаемое значение: логическое значение bool - успешность установки нового имени.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода set_custom_name класса cl_custom_base

№	Предикат	Действия	№ перехода
1	не является ли родительский объект get_main_custom() текущего объекта нулевым указателем и существует ли подпользовательский объект с заданным именем new_custom_name для родительского класса?	возврат логического значения 'false'	2
			2
2		значение 'custom_name' текущего объекта класса устанавливается равным значению 'new_custom_name'	3
3		возврат логического значения 'true'	∅

3.5 Алгоритм метода get_main_custom класса cl_custom_base

Функционал: возврат указателя на родительский пользовательский объект текущего объекта.

Параметры: отсутствуют.

Возвращаемое значение: p_main_custom_object - указатель на родительский пользовательский объект.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_main_custom класса cl_custom_base

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

3.6 Алгоритм метода get_sub_custom класса cl_custom_base

Функционал: возврат указателя на подпользовательский объект текущего объекта по его имени.

Параметры: string custom_name - имя подпользовательского объекта.

Возвращаемое значение: p_sub_customs - указатель на подпользовательский объект.

Алгоритм метода представлен в таблице 7.

Таблица 7 – 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	совпадает ли имя вектора p_sub_customs с заданным именем custom_name?	метод 'get_sub_custom()' возвращает указатель на подпользовательский объект с индексом 'i' из вектора p_sub_customs	∅
			2
4		возврат nullptr	∅

3.7 Алгоритм метода print_custom_tree класса cl_custom_base

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

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода print_custom_tree класса cl_custom_base

№	Предикат	Действия	№ перехода
1	больше ли размер вектора p_sub_customs 0?	переход на следующую строку, вывод имени текущего пользовательского объекта на экран	2
			∅
2		инициализация целочисленной переменной i	3
3	меньше ли переменная i количества элементов в векторе p_sub_customs?	i++; вывод имени подпользовательского объекта с индексом 'i' из вектора 'p_sub_customs' на экран	4
			∅
4		вызов метода 'print_custom_tree()' для подпользовательского объекта с индексом 'i' из вектора 'p_sub_customs'	∅

3.8 Алгоритм функции main

Функционал: основной алгоритм работы программы.

Параметры: отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения работы программы.

Алгоритм функции представлен в таблице 9.

Таблица 9 – 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	∅

3.9 Алгоритм конструктора класса cl_custom_1

Функционал: конструктор.

Параметры: cl_custom_base* p_main - указатель на родительский класс, string custom_name - имя пользовательского объекта.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_custom_1

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_1', который наследуется от класса 'cl_custom_base', конструктор принимает указатель на родительский объект типа 'cl_custom_base*' и строку 'custom_name', затем передает их конструктору базового класса 'cl_custom_base' для инициализации	∅

3.10 Алгоритм конструктора класса cl_custom_application

Функционал: инициализация объекта приложения, используя переданный родительский объект.

Параметры: p_main .

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_custom_application

№	Предикат	Действия	№ перехода
1		определение конструктора класса 'cl_custom_application', который наследует от класса 'cl_custom_base', конструктор принимает указатель на родительский класса типа 'cl_custom_base*' и передает его конструктору базового класса для инициализации	∅

3.11 Алгоритм метода build_tree_objects класса cl_custom_application

Функционал: построение иерархии пользовательских объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода build_tree_objects класса cl_custom_application

№	Предикат	Действия	№ перехода
1		инициализация переменных main_custom_name, sub_custom_name типа 'string'	2
2		создание указателя 'p_main' на объект класса 'cl_custom_base' и инициализация его значением указателя на текущий объект 'this'	3
3		объявление указателя 'p_main' на объект класса 'cl_custom_base' и инициализация его значением 'nullptr' указывающим на нулевой адрес	4
4		ввод значения переменной main_custom_name с клавиатуры	5
5		вызов метода 'set_custom_name()' в качестве аргумента	6

№	Предикат	Действия	№ перехода
6	истинно?	ввод значения переменных main_custom_name, sub_custom_name с клавиатуры	7
			∅
7	равны ли main_custom_name и sub_custom_name?	прерывание выполнения цикла	8
			8
8	сущесвует ли указатель на объект и равно ли имя текущего объекта имени объекта?	присваивание значение переменной 'p_sub' переменной 'p_main'	9
			9
9	не существует ли для 'p_main' 'sub_custom_name' и 'main_custom_name' равно имени объекта 'p_main'?	: создание нового объекта класса 'cl_custom_1' с родительским объектом 'p_main' и именем 'sub_custom_name', присвоение указателя на этот новый объект переменной 'p_sub'	∅
			∅

3.12 Алгоритм метода exec_custom_app класса cl_custom_application

Функционал: выполнение пользовательского приложения.

Параметры: отсутствуют.

Возвращаемое значение: int - индикатор корректности работы.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода exec_custom_app класса cl_custom_application

№	Предикат	Действия	№ перехода
1		вывод имени текущего пользовательского объекта	2
2		вызов метода 'print_custom_tree()', который выводит иерархию	∅

№	Предикат	Действия	№ перехода
		пользовательских объектов	

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-9.

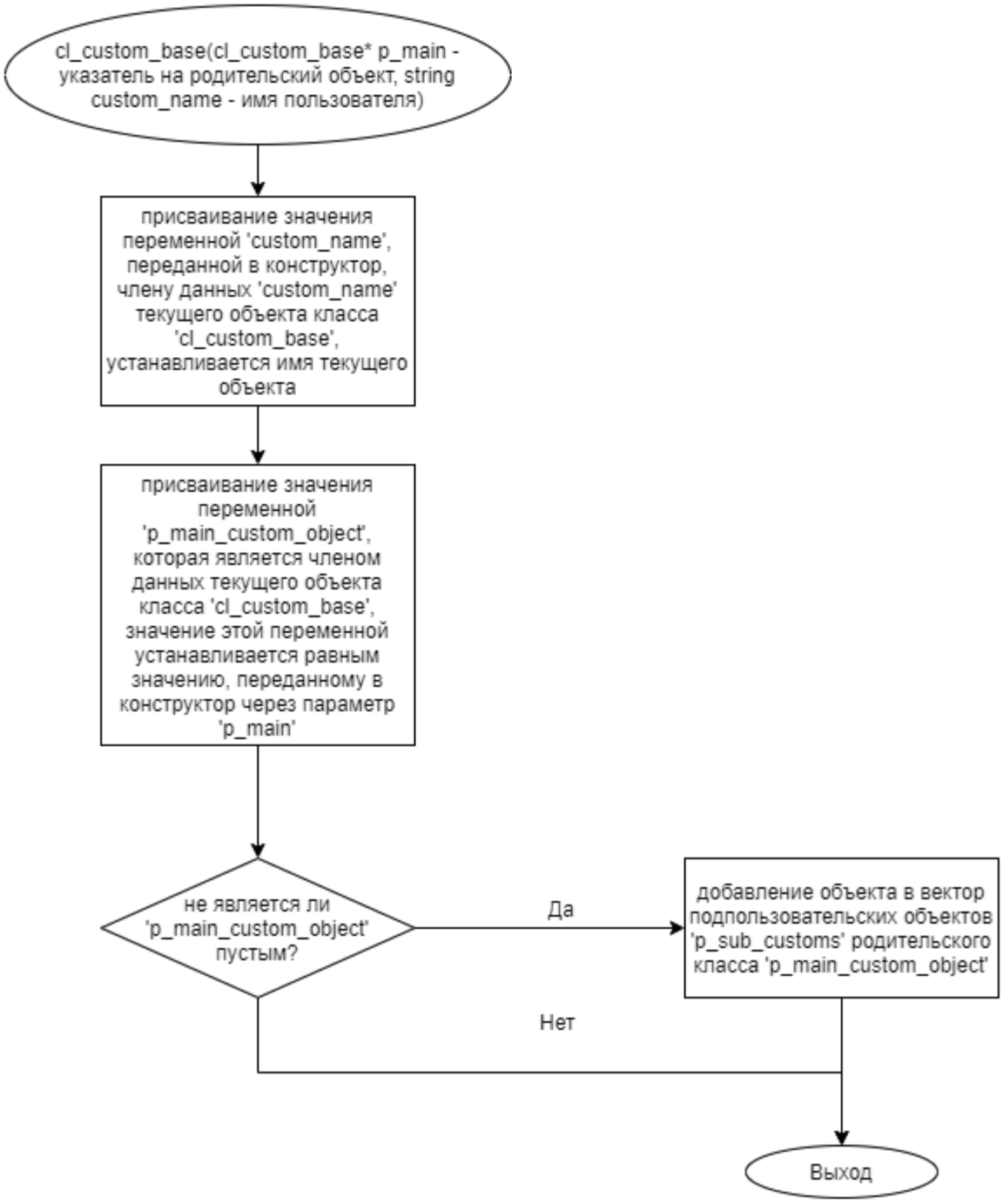

Рисунок 1 – Блок-схема алгоритма

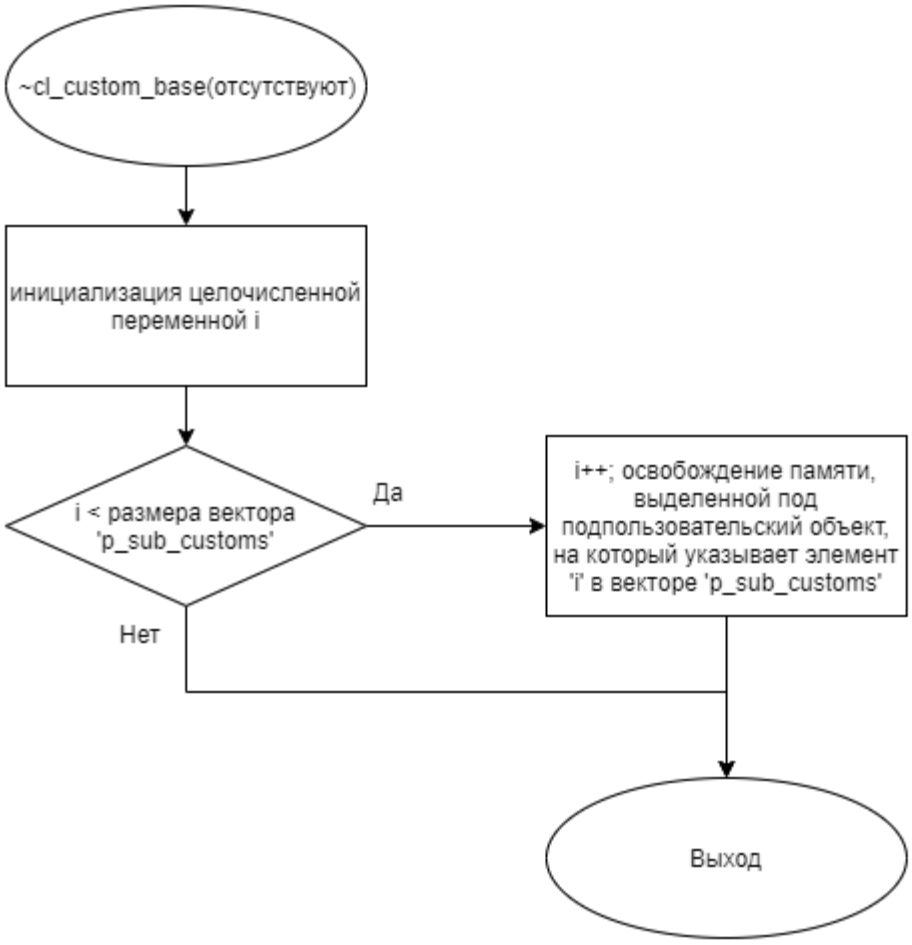

Рисунок 2 – Блок-схема алгоритма

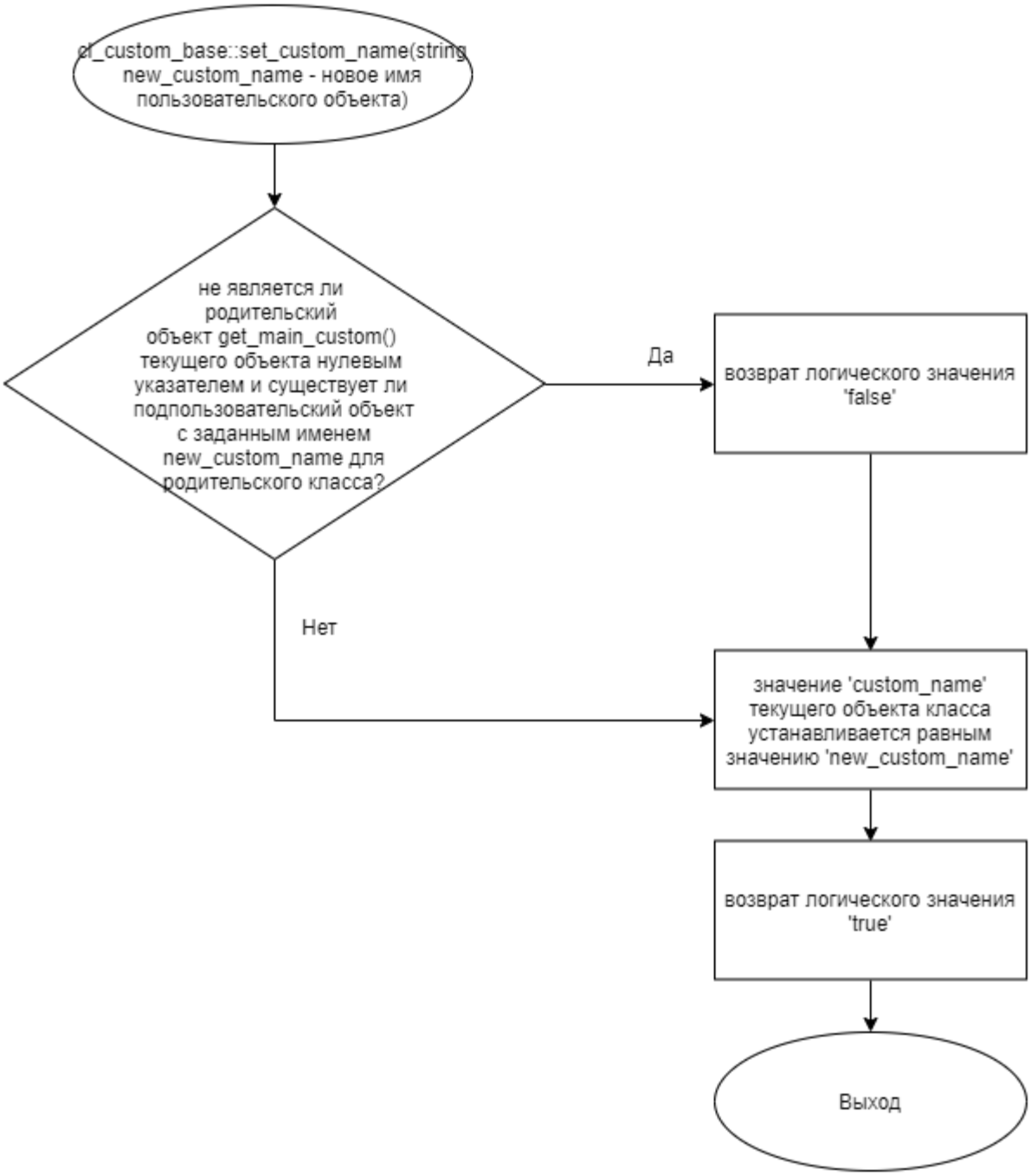

Рисунок 3 – Блок-схема алгоритма

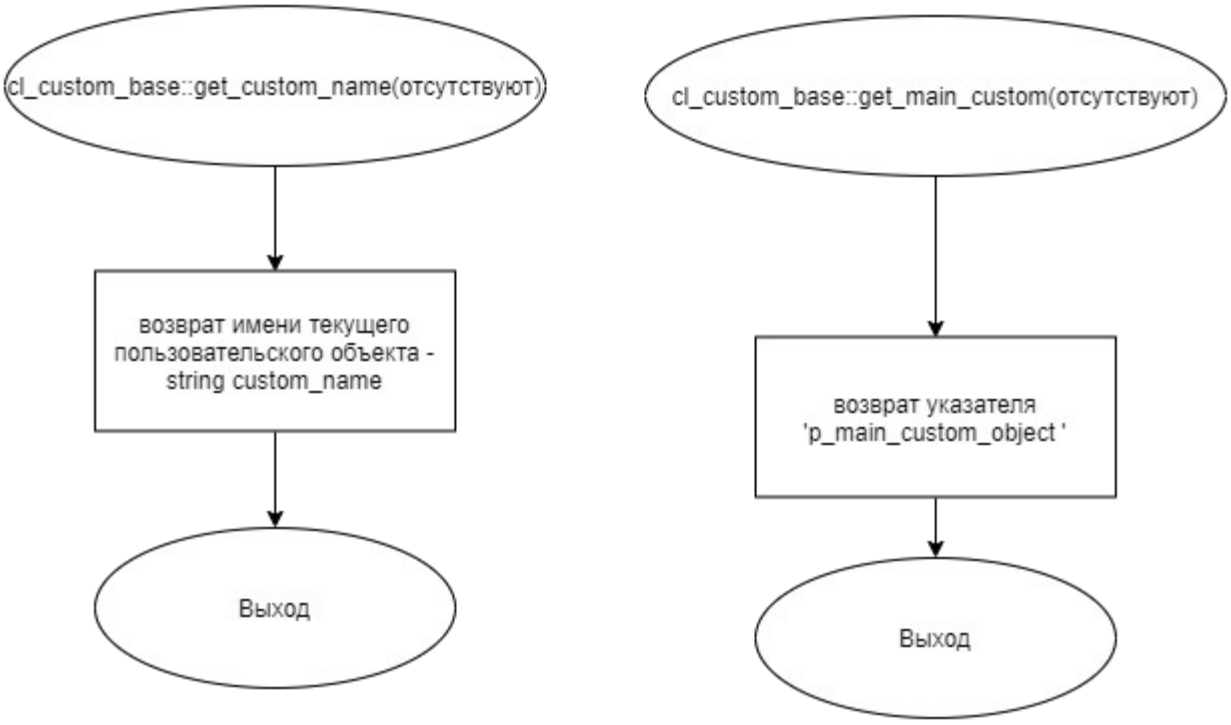

Рисунок 4 – Блок-схема алгоритма

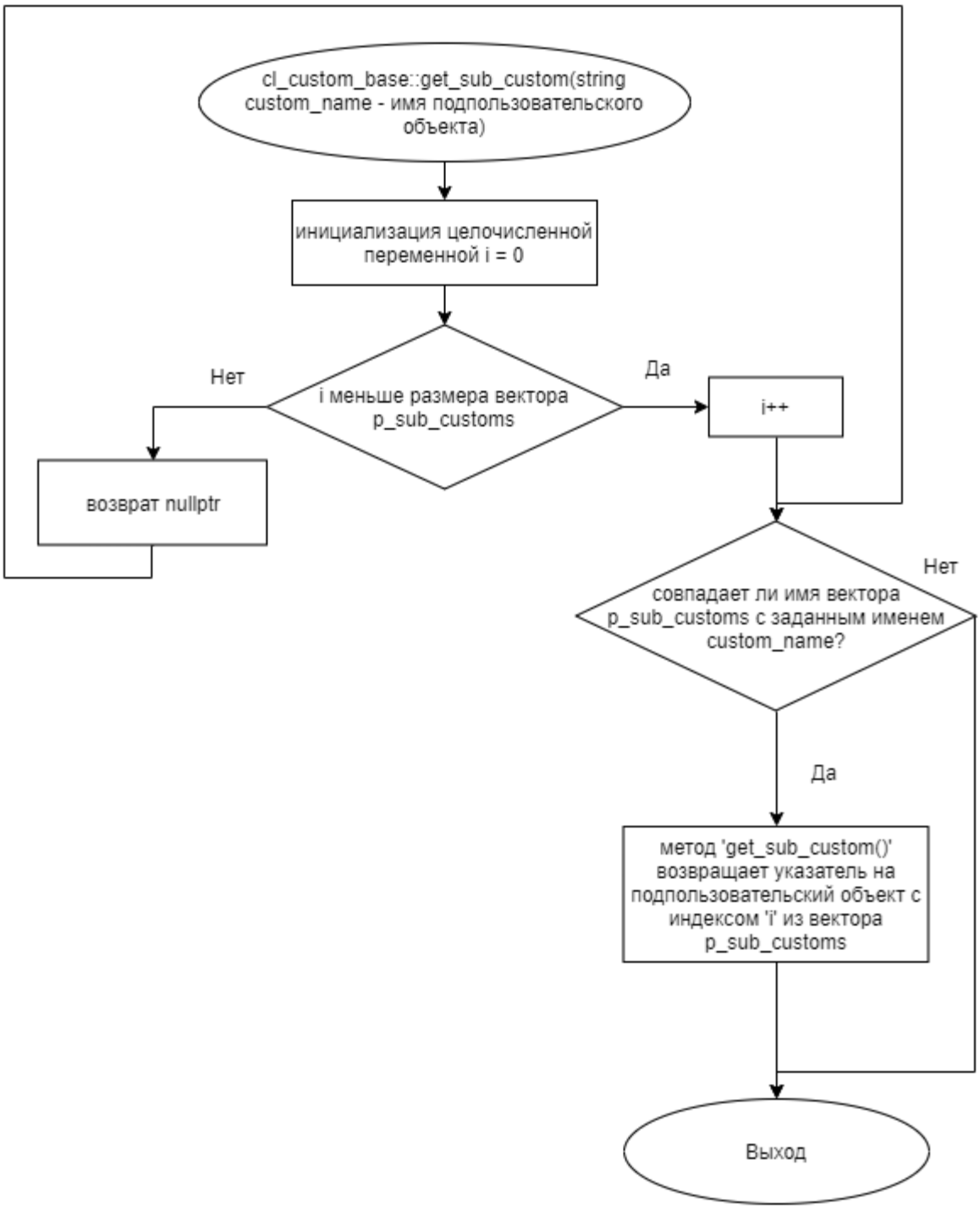

Рисунок 5 – Блок-схема алгоритма

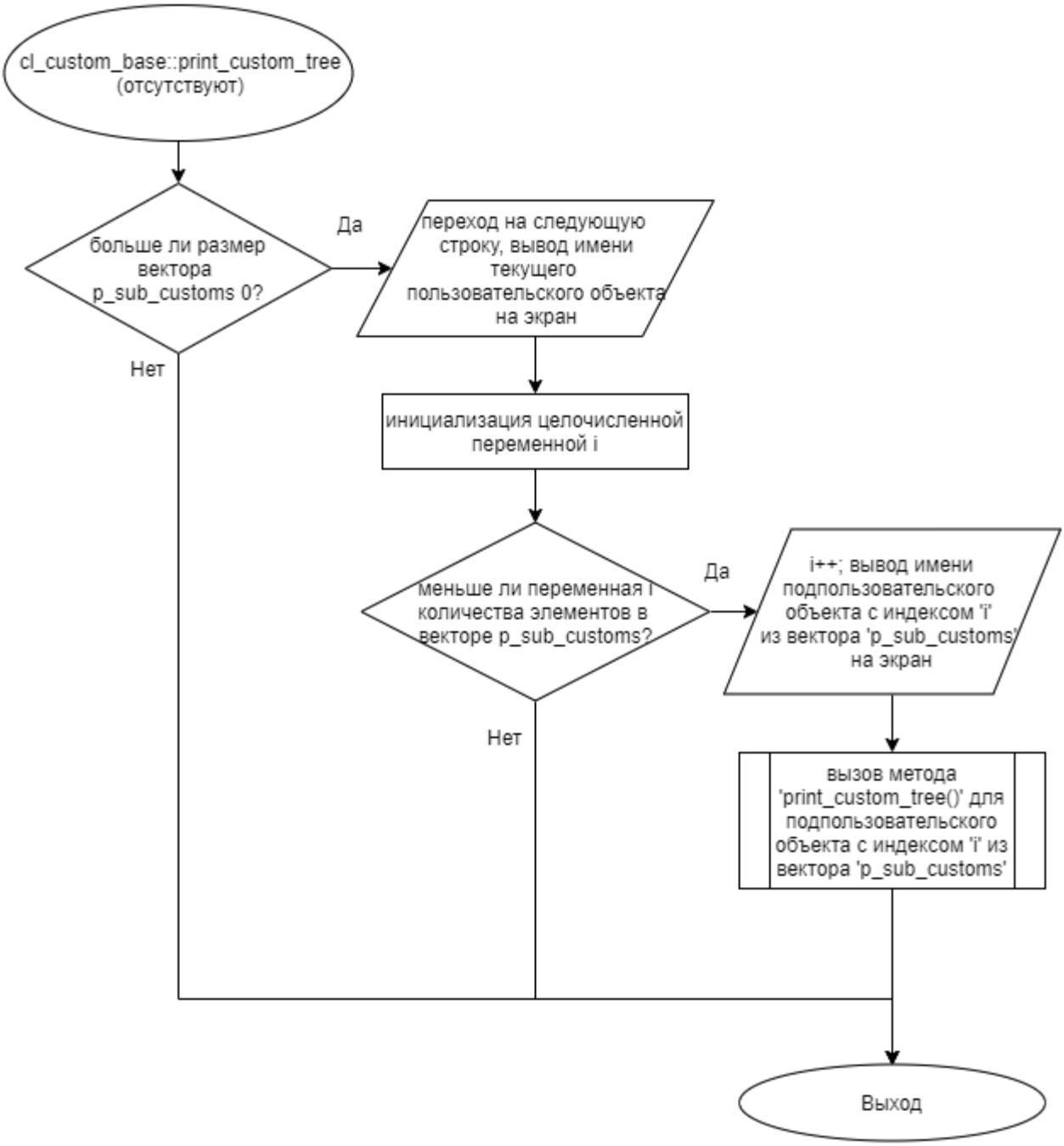

Рисунок 6 – Блок-схема алгоритма

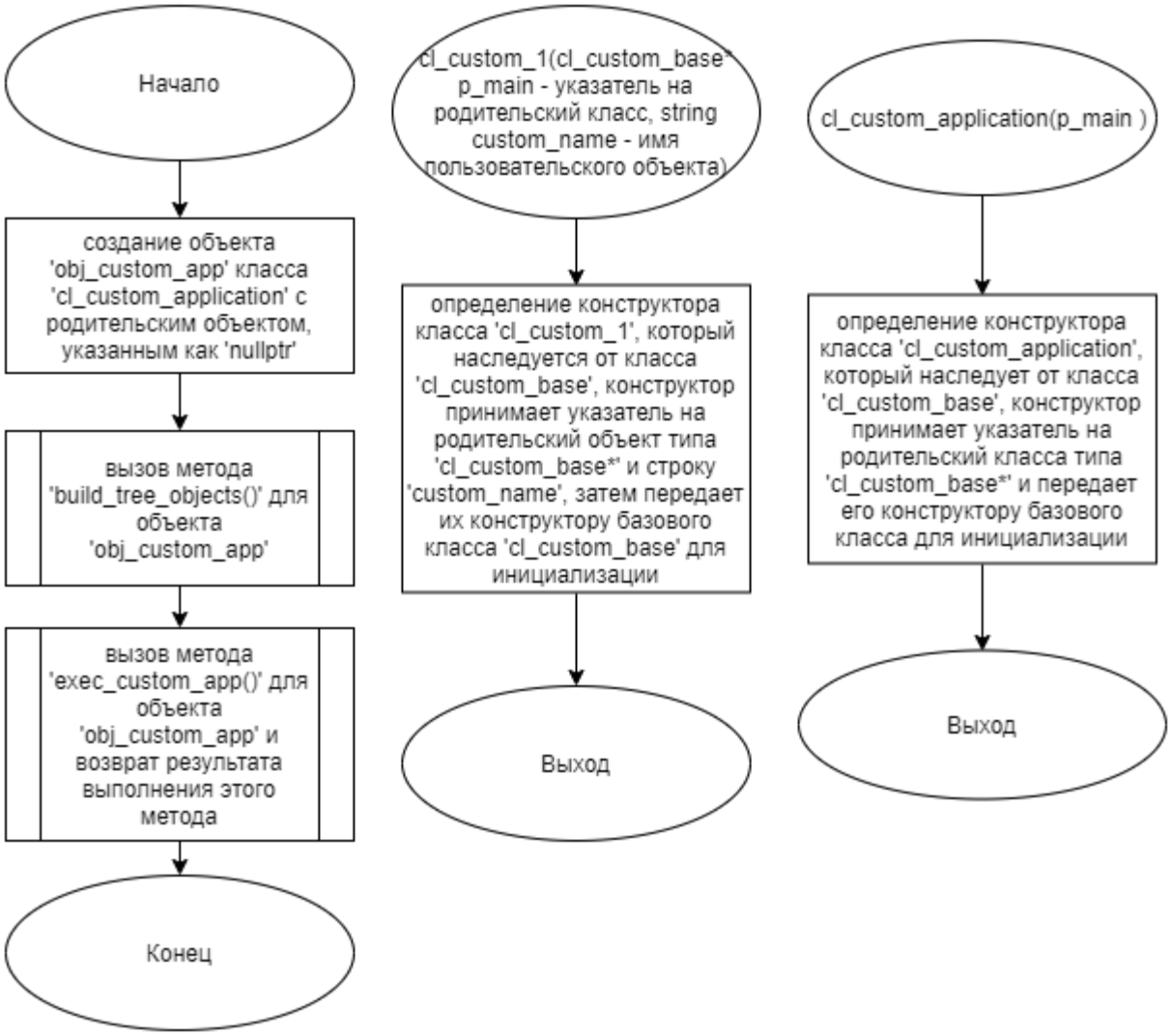

Рисунок 7 – Блок-схема алгоритма

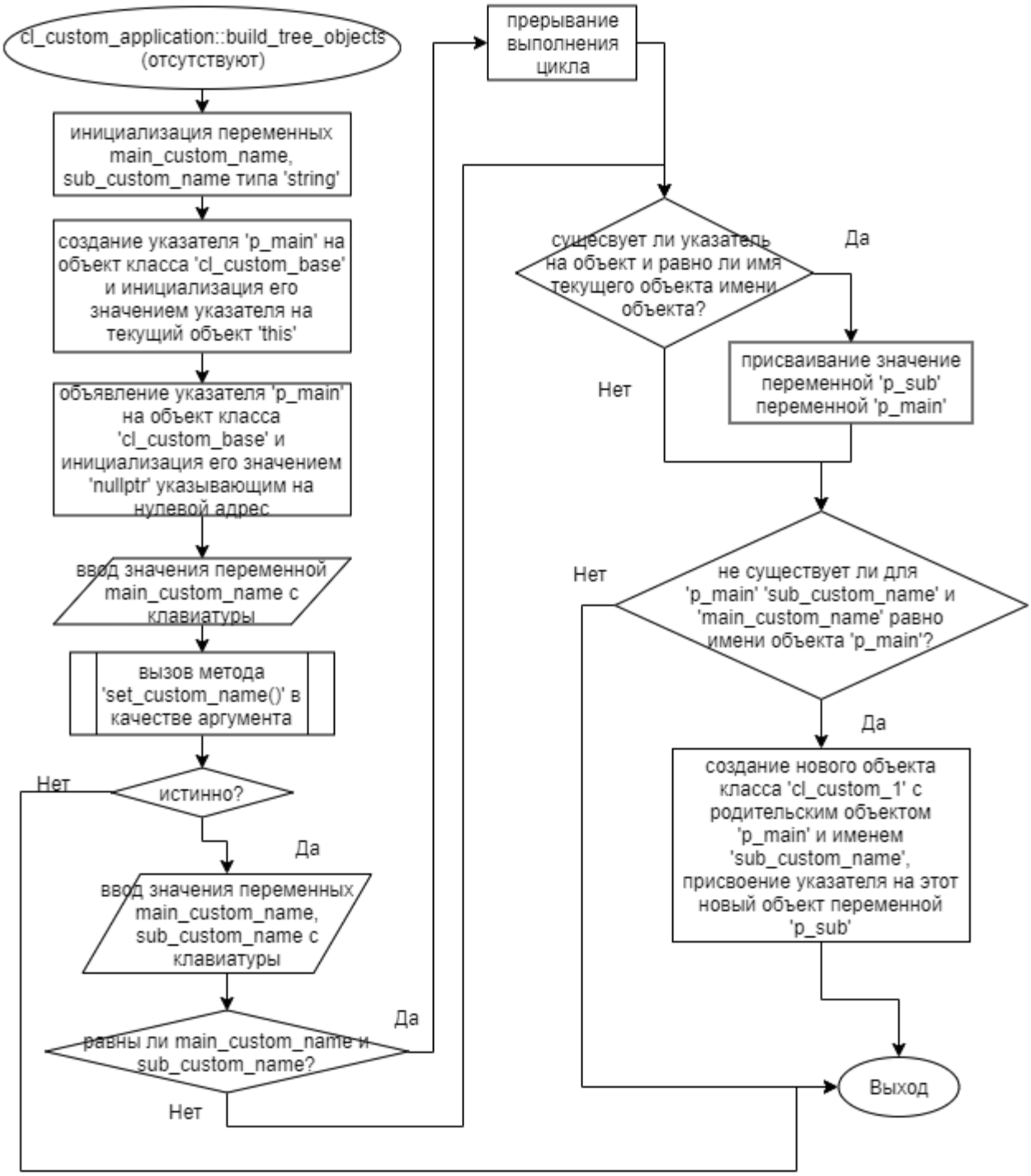

Рисунок 8 – Блок-схема алгоритма

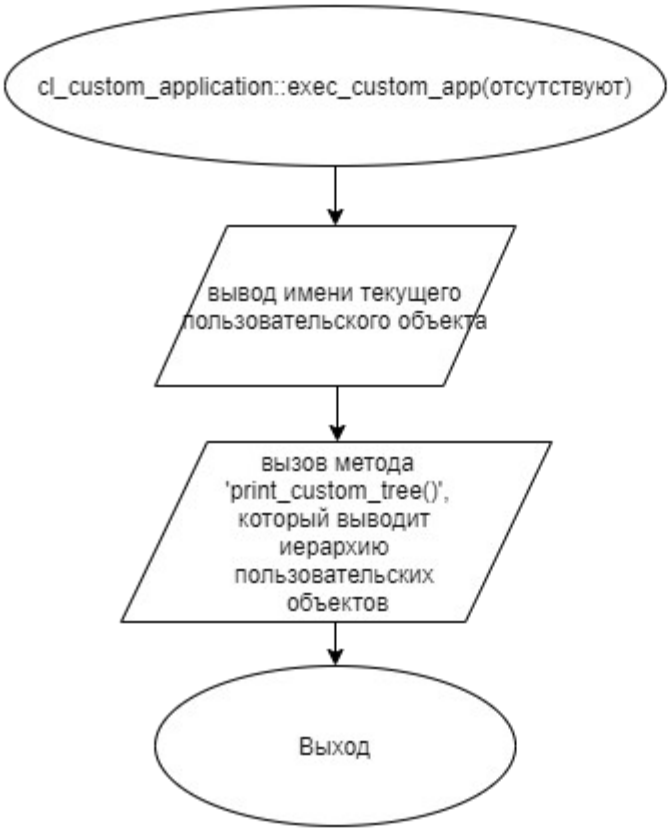

Рисунок 9 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_custom_1.cpp

Листинг 1 – cl_custom_1.cpp

```
#include "cl_custom_1.h"
#include <iostream>

cl_custom_1::cl_custom_1(cl_custom_base*   p_main,   string   custom_name)   :
cl_custom_base(p_main, custom_name) {}
```

5.2 Файл cl_custom_1.h

Листинг 2 – cl_custom_1.h

```
#ifndef __CL_CUSTOM_1__H
#define __CL_CUSTOM_1__H

#include "cl_custom_base.h"
#include <iostream>

class cl_custom_1 : public cl_custom_base {
public:
   cl_custom_1(cl_custom_base* p_main, string custom_name);
};
#endif
```

5.3 Файл cl_custom_application.cpp

Листинг 3 – cl_custom_application.cpp

```
#include "cl_custom_application.h"
#include "cl_custom_1.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);

   while(true) {
      cin >> main_custom_name >> sub_custom_name;
      if(main_custom_name == sub_custom_name) {
         break;
      }
      if(p_sub != nullptr && main_custom_name == p_sub->get_custom_name()) {
         p_main = p_sub;
      }
      if(p_main->get_sub_custom(sub_custom_name)      ==      nullptr      &&
main_custom_name == p_main->get_custom_name()) {
         p_sub = new cl_custom_1(p_main, sub_custom_name);
      }
   }
}

int cl_custom_application::exec_custom_app() {
   cout << get_custom_name();
   print_custom_tree();
   return 0;
}
```

5.4 Файл cl_custom_application.h

Листинг 4 – 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 {
public:
   cl_custom_application(cl_custom_base* p_main);
   void build_tree_objects();
   int exec_custom_app();
};
#endif
```

5.5 Файл cl_custom_base.cpp

Листинг 5 – cl_custom_base.cpp

```
#include "cl_custom_base.h"
#include "cl_custom_1.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];
   }
};

string cl_custom_base::get_custom_name() {
   return custom_name;
}

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      &&      get_main_custom()-
>get_sub_custom(new_custom_name) != nullptr) {
      return false;
   }
   custom_name = new_custom_name;
   return true;
};

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]->custom_name == custom_name) {
         return p_sub_customs[i];
      }
   };

   return nullptr;
};

void cl_custom_base::print_custom_tree() {
   if(p_sub_customs.size() > 0) {
```

```
        cout << endl << custom_name;
        for(int i = 0; i < p_sub_customs.size(); i++) {
           cout << "  " << p_sub_customs[i]->get_custom_name();
           p_sub_customs[i]->print_custom_tree();
        }
     };
};
```

5.6 Файл cl_custom_base.h

Листинг 6 – cl_custom_base.h

```
#ifndef __CL_CUSTOM_BASE__H
#define __CL_CUSTOM_BASE__H
#include <vector>
#include <iostream>

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;

public:
   cl_custom_base(cl_custom_base*   p_main,   string   custom_name   =   "Base
Objcet");
   ~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);
   void print_custom_tree();
};
#endif
```

5.7 Файл main.cpp

Листинг 7 – 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 14.

Таблица 14 – 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_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

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).