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 используются:

1. Операторы ввода/вывода cin/cout.
2. Встроенные структуры данных, такие как std::set, std::vector и std::string.
3. Условный оператор if .
4. Операторы цикла for и while.
5. **Класс cl_base**, содержащий**:**
    5.1. Следующие **свойства** с модификатором доступа private:
        5.1.1. Указатель на головной объект.
        5.1.2. Имя данного объекта строкового типа.
        5.1.3. Массив указателей на подчинённые объекты.
    5.2. Следующие **методы** с модификатором доступа public:
        5.2.1. Параметризованный конструктор cl_base(). Функционал - создание объекта класса cl_base и инициализация закрытых свойств этого обьекта.
        5.2.2. Деструктор ~cl_base(). Функционал - освобождение памяти занимаемой обьектом.
        5.2.3. Метод set_name(). Функционал - редактирование имени обьекта.
        5.2.4. Метод get_name(). Функционал - получение имени данного обьекта.
        5.2.5. Метод print_tree(). Функционал - вывод имен объектов дерева иерархии в консоль.
        5.2.6. Метод get_head_ptr(). Функционал - получение указателя на головной обьект.
        5.2.7. Метод get_sub_object(). Функционал - получение

указателя на подчиненный обьект по его имени. Если объекта с заданным именем не существует, возвращает пустой указатель.

6. **Класс cl_application**, наследованный от класса cl_base. Имеет следующие методы:

   6.1. Параметризованный конструктор cl_application(). Функционал - создание объекта класса cl_application и инициализация его закрытых свойств..

   6.2. Метод build_tree_objects(). Функционал - создание дерева иерархии объектов.

   6.3. Метод exec_app(). Функционал - запуск вывода имён объектов дерева иерархии.

7. **Класс cl_2**, наследованный от cl_base.

   7.1. Параметризованный конструктор cl_2(). Функционал - создание объекта класса cl_2 и инициализация его закрытых свойств.

8. **Класс cl_3**, наследованный от cl_base.

   8.1. Параметризованный конструктор cl_3(). Функционал - создание объекта класса cl_3 и инициализация закрытых свойств обьект.

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: Инициализация обьекта класса.

Параметры: ссылочный тип на cl_base p_head_obj - ссылка на головной обьект, строковой тип self_name - имя обьекта.

Алгоритм конструктора представлен в таблице 1.

Таблица 1 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		p_head_object = p_head_obj	2
2		name = self_name	3
3	p_head_object	p_head_object->p_sub_objects.push_back(this)	∅
			∅

3.2 Алгоритм деструктора класса cl_base

Функционал: Уничтожение из памяти всех подчиненных обьектов.

Параметры: Отсутствуют.

Алгоритм деструктора представлен в таблице 2.

Таблица 2 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		i = 0	2
2	i < p_sub_objects.size()	Удаление из памяти i-го подчиненного обьекта	2

№	Предикат	Действия	№ перехода
		функцией delete. i += 1	
			∅

3.3 Алгоритм метода set_name класса cl_base

Функционал: изменение имени обьекта.

Параметры: строковой тип new_name.

Возвращаемое значение: булевый тип.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода set_name класса cl_base

№	Предикат	Действия	№ перехода
1		Обьявление множества copies	2
2	не p_head_object	name = new_name. Возвращает булевое значение true	∅
			3
3		i = 0	4
4	i < количество подчиненных обьектов головного обьекта	i += 1	5
			6
5	Имя i-го обьекта входит в множество copies	Возвращает булевое значение false	∅
		Добавляет имя i-го обьекта в множество copies	4
6		name = new_name	7
7		Возвращает булевое значение true	∅

3.4 Алгоритм метода get_name класса cl_base

Функционал: Возвращает имя обьекта.

Параметры: Отсутствуют.

Возвращаемое значение: Строковой тип - имя обьекта.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода
1		Возвращает строковое значение name	∅

3.5 Алгоритм метода get_head_ptr класса cl_base

Функционал: Возвращение ссылки на головной обьект.

Параметры: Отсутствуют.

Возвращаемое значение: ссылка на cl_base.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_head_ptr класса cl_base

№	Предикат	Действия	№ перехода
1		возвращает ссылку на головной обьект p_head_object	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-10.

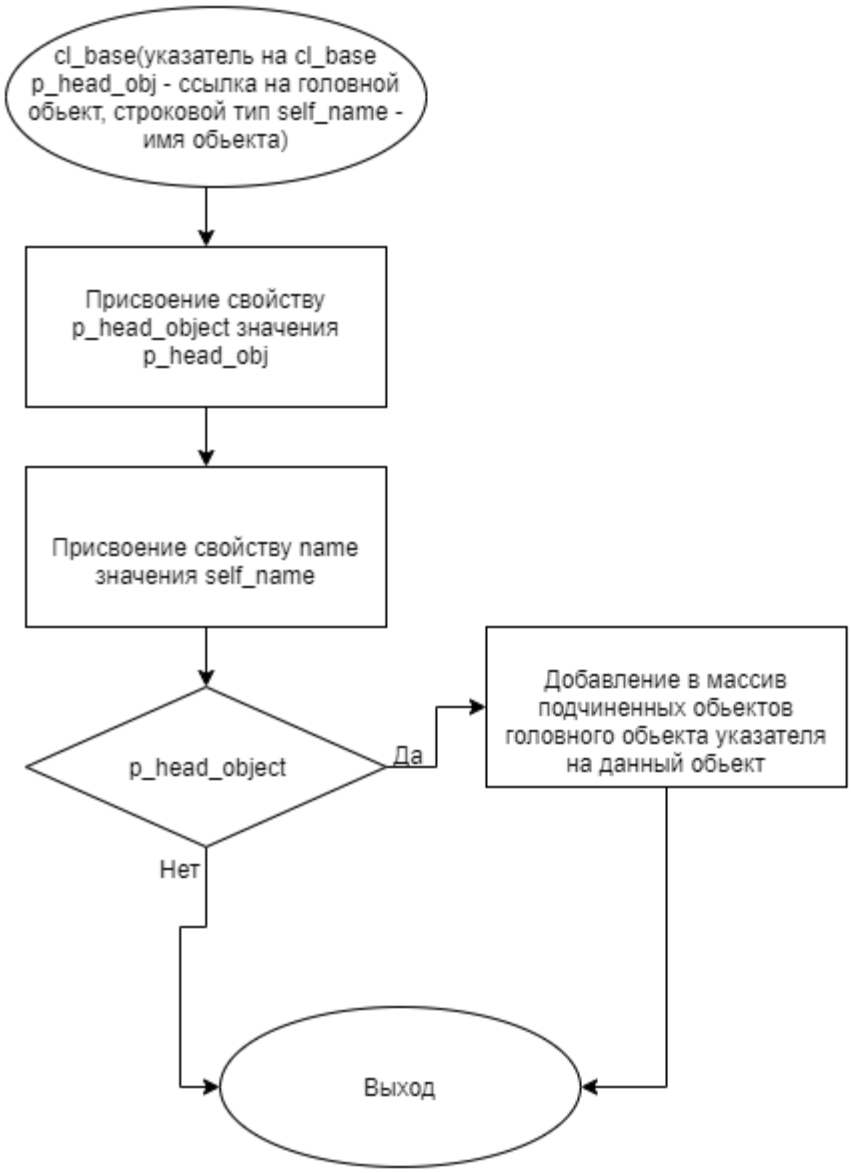

Рисунок 1 – Блок-схема алгоритма

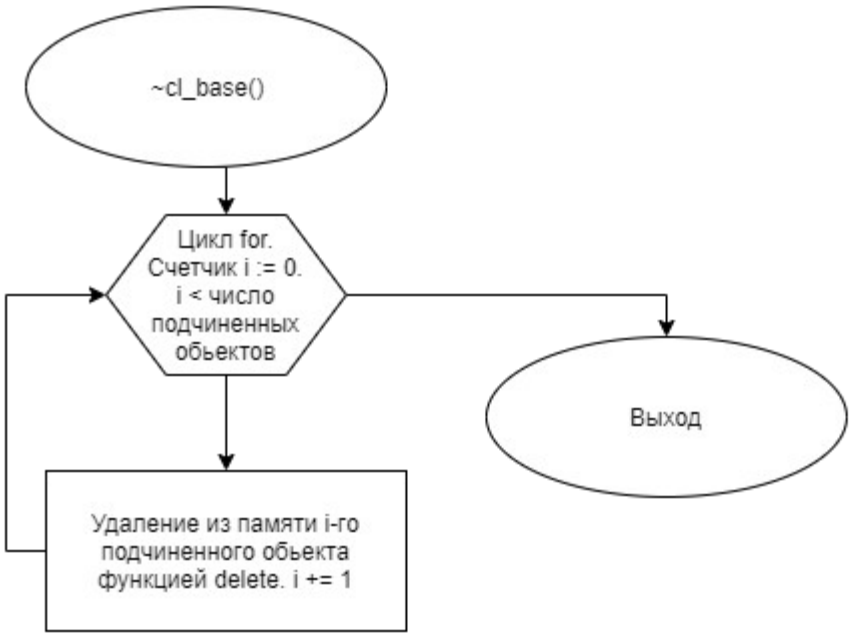

Рисунок 2 – Блок-схема алгоритма

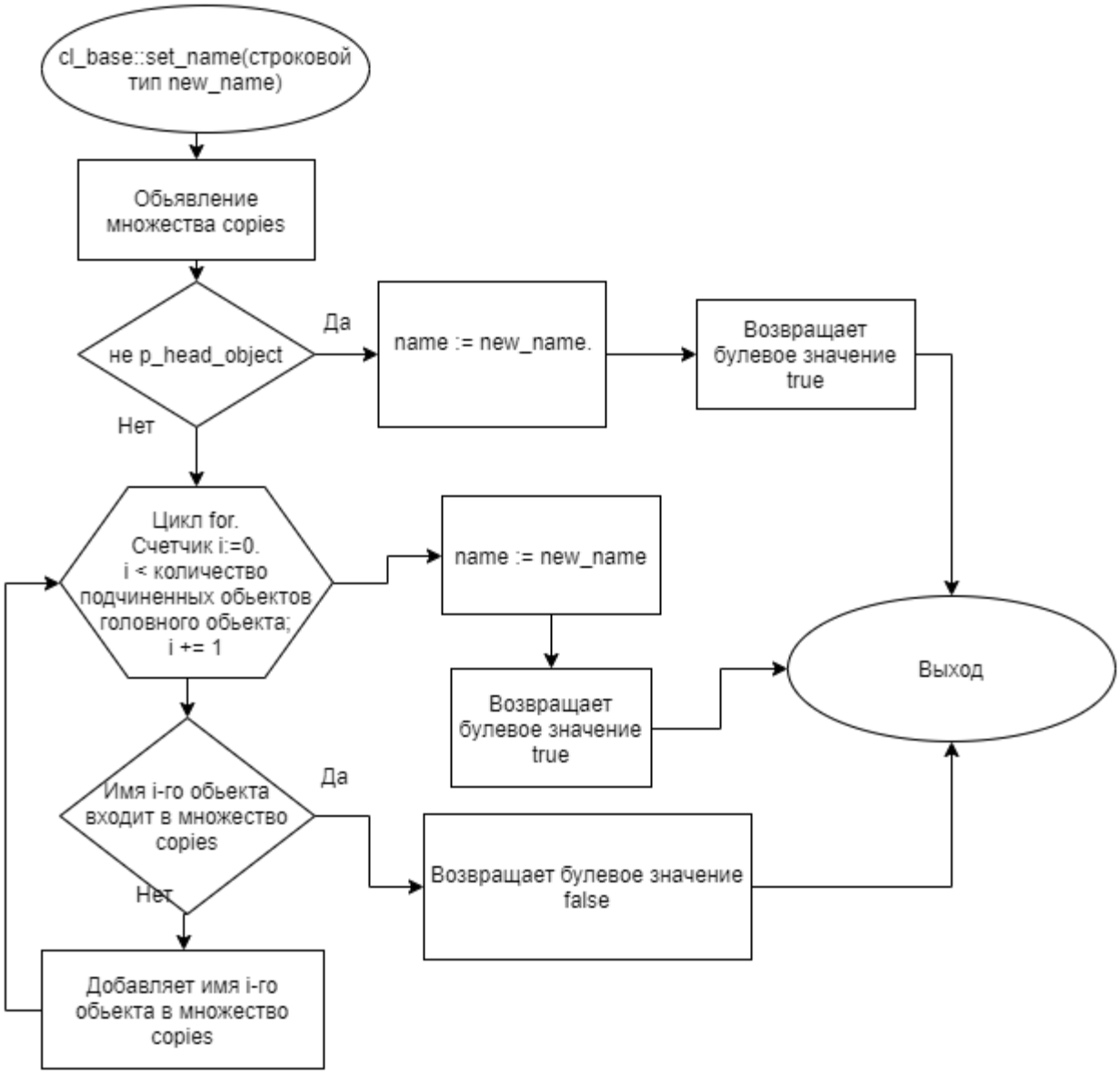

Рисунок 3 – Блок-схема алгоритма

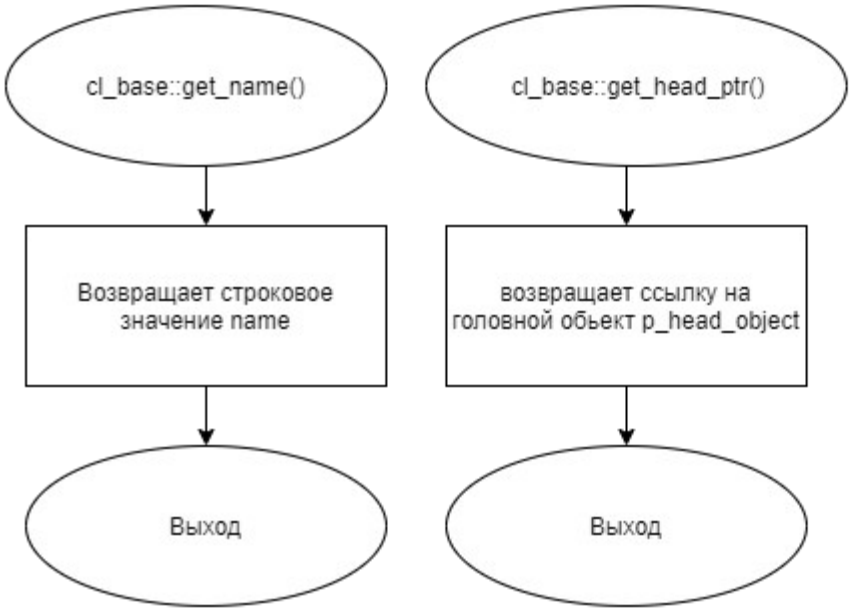

Рисунок 4 – Блок-схема алгоритма

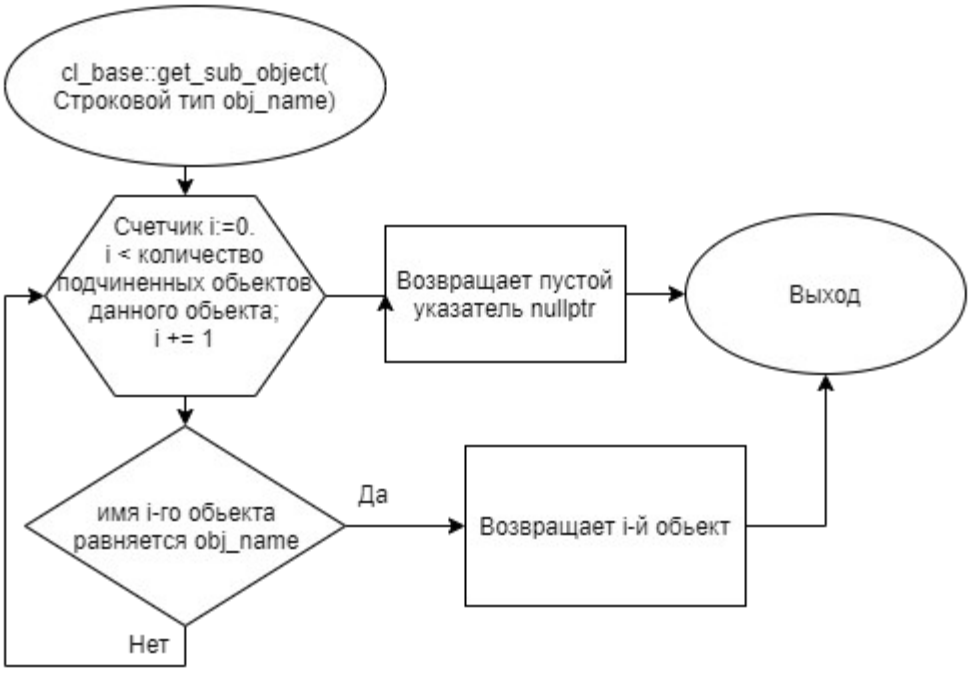

Рисунок 5 – Блок-схема алгоритма

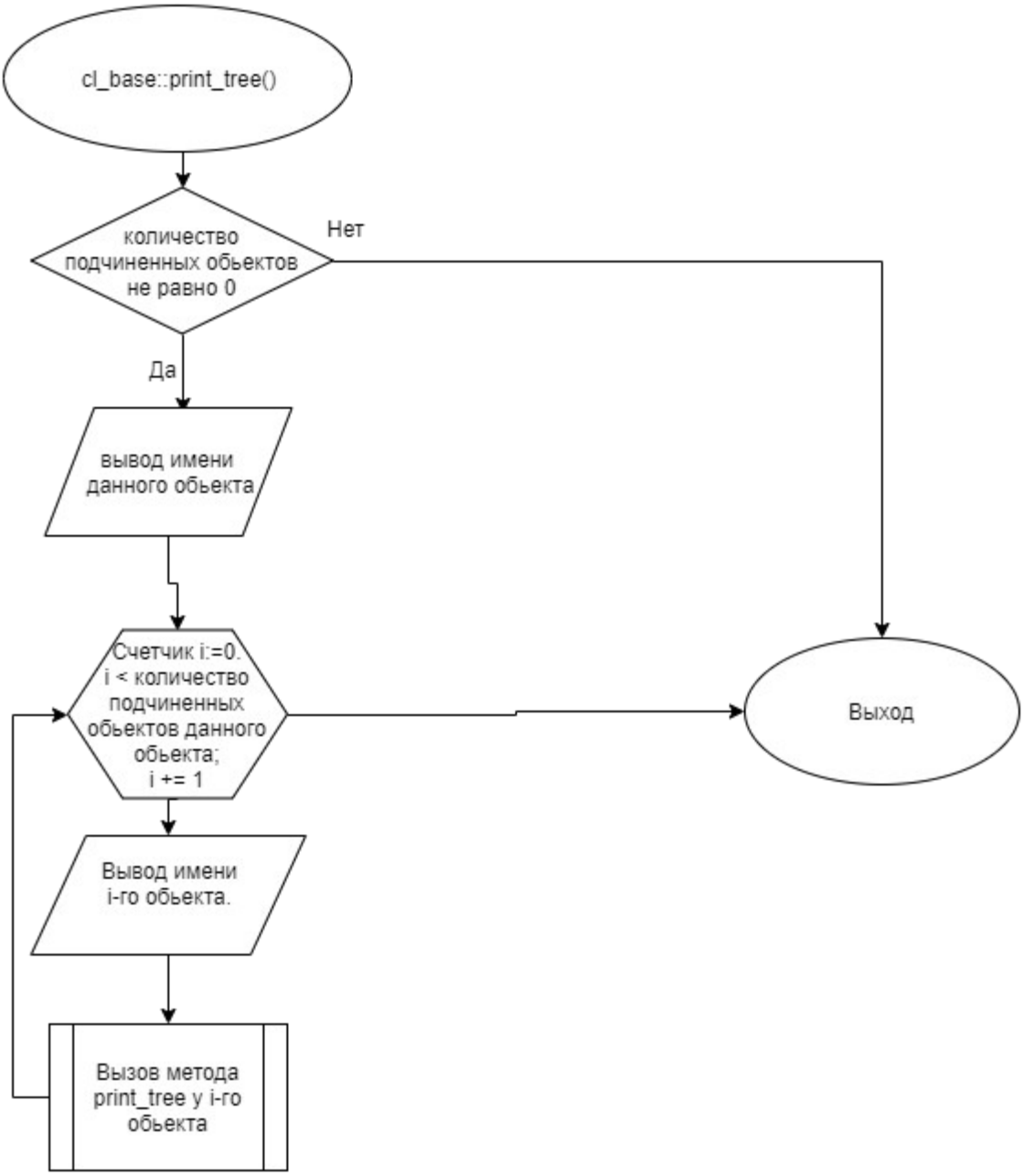

Рисунок 6 – Блок-схема алгоритма

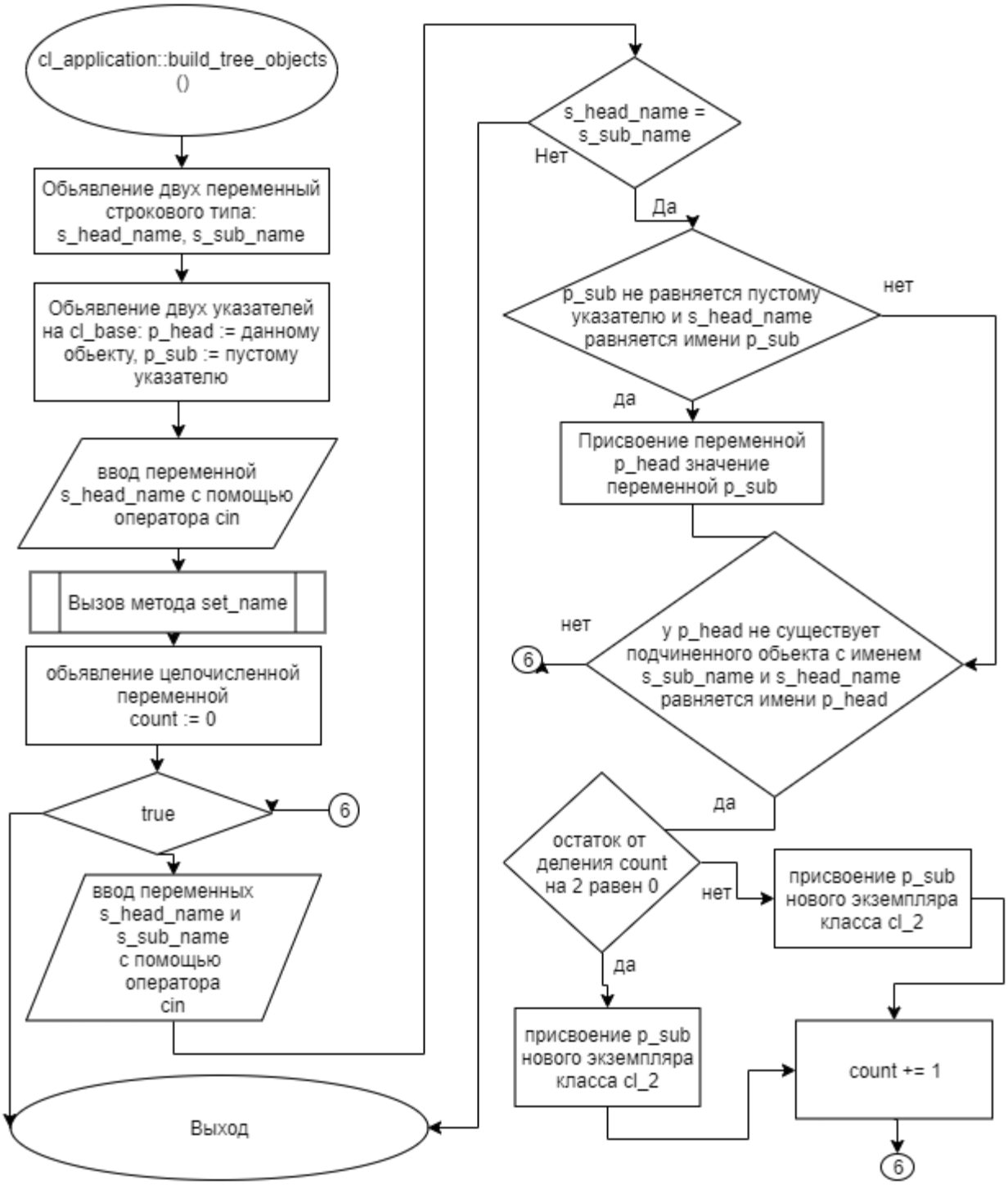

Рисунок 7 – Блок-схема алгоритма

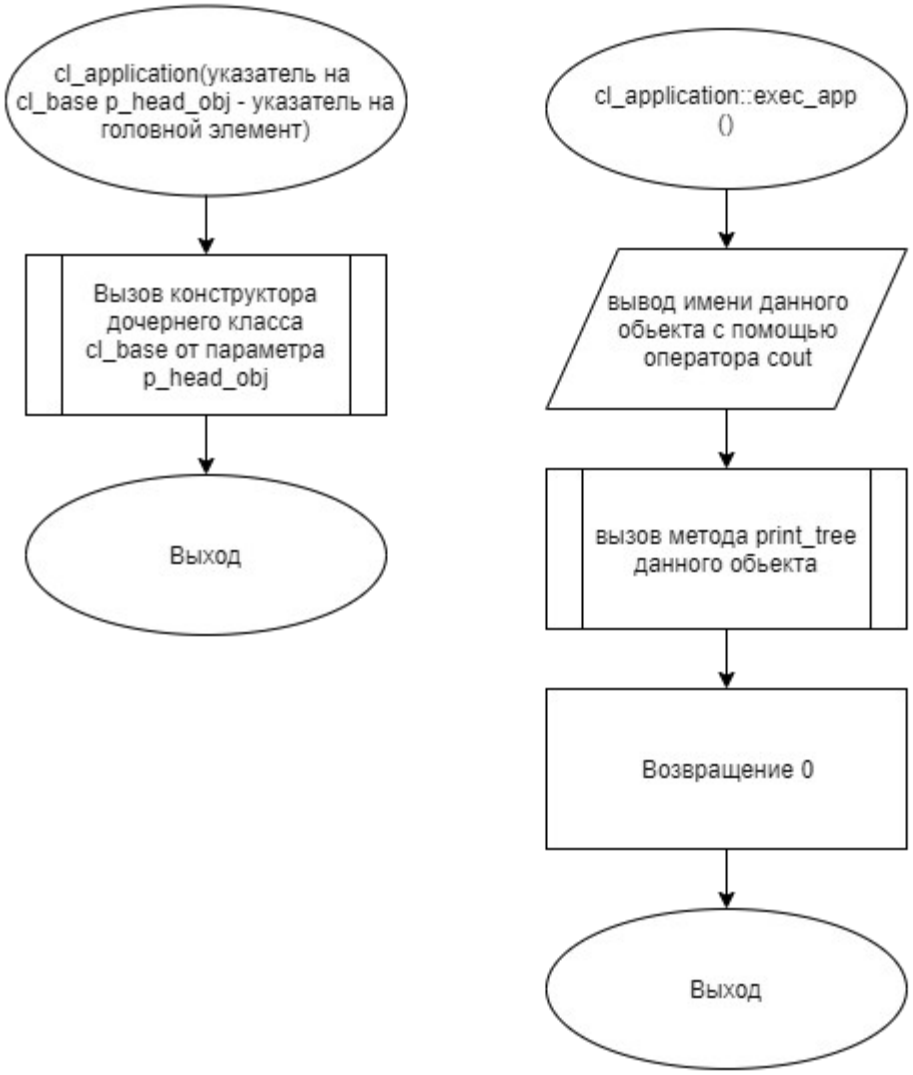

Рисунок 8 – Блок-схема алгоритма

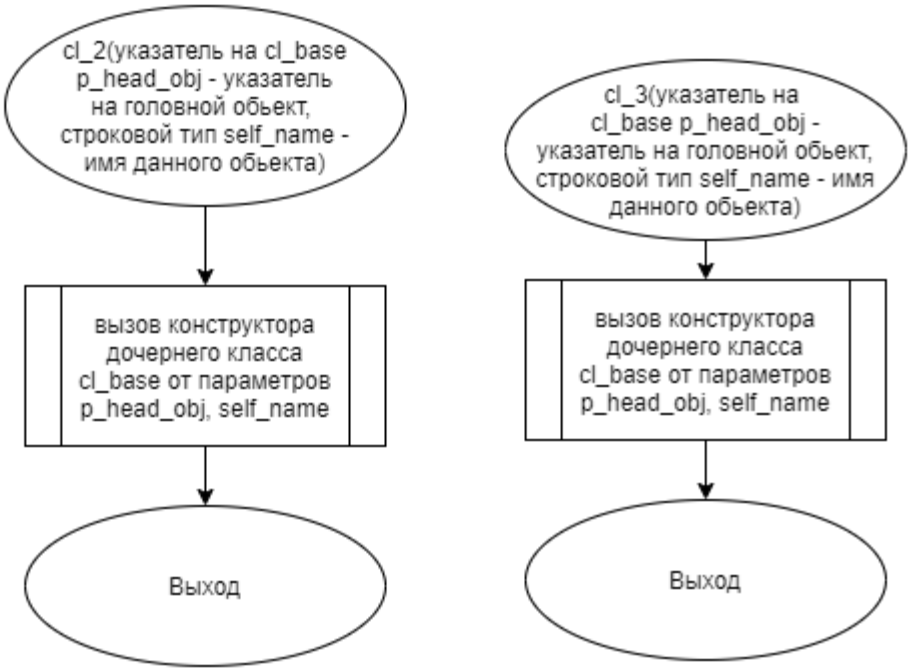

Рисунок 9 – Блок-схема алгоритма

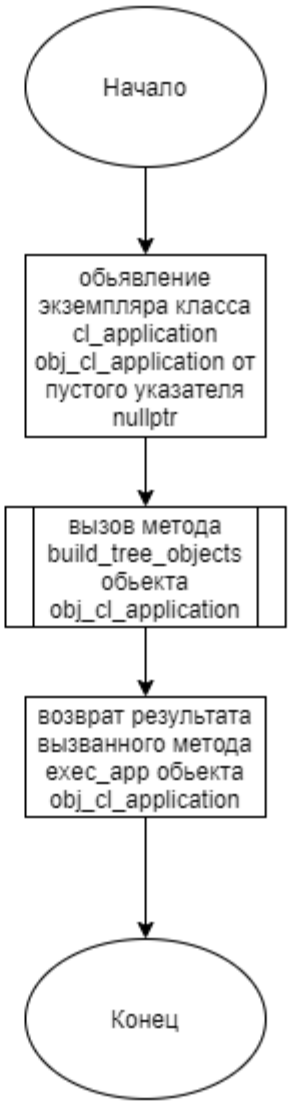

Рисунок 10 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.hpp"

cl_2::cl_2(cl_base*    p_head_obj,    string    self_name):cl_base(p_head_obj,
self_name){
}
```

5.2 Файл cl_2.hpp

Листинг 2 – cl_2.hpp

```
#ifndef __CL_2__HPP
#define __CL_2__HPP
#include "cl_base.hpp"

class cl_2:public cl_base{
public:
   cl_2(cl_base* p_head_obj, string self_name);
};


#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.hpp"
#include <string>

cl_3::cl_3(cl_base*    p_head_obj,    string    self_name):cl_base(p_head_obj,
```

```
self_name){
}
```

5.4 Файл cl_3.hpp

Листинг 4 – cl_3.hpp

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.hpp"

class cl_3:public cl_base{
public:
   cl_3(cl_base* p_head_obj, string self_name);
};

#endif
```

5.5 Файл cl_application.cpp

Листинг 5 – cl_application.cpp

```
#include "cl_application.hpp"
#include "cl_base.hpp"
#include "cl_2.hpp"
#include "cl_3.hpp"
#include <iostream>
#include <string>
using namespace std;

cl_application::cl_application(cl_base* p_head_obj):cl_base(p_head_obj){
}

void cl_application::build_tree_objects(){
   string s_head_name, s_sub_name;
   cl_base *p_sub = nullptr;
   cl_base *p_head = this;
   cin >> s_head_name;
   set_name(s_head_name);
   int count = 0;
   while (true){
      cin >> s_head_name >> s_sub_name;
      //1. end of loop condition
      if (s_head_name == s_sub_name){
         break;
```

```
        }
        //2. creating a new branch, sub_name should match p_sub name, p_head
    becomes a cl_3 object
        if (p_sub != nullptr && p_sub->get_name() == s_head_name){
           p_head = p_sub;
           //p_sub = new cl_base(p_head, s_sub_name);
        }
        //3. adding an object of cl_2 class to an existing branch
        if(p_head->get_sub_object(s_sub_name)  ==  nullptr  &&  s_head_name  ==
    p_head->get_name()){
           if (count % 2 == 0){
              p_sub = new cl_2(p_head, s_sub_name);
           }
           else{
              p_sub = new cl_3(p_head, s_sub_name);
           }
           count ++;
        }

     }
    }

    int cl_application::exec_app(){
       cout << get_name();
       print_tree();
       return 0;
    }
```

5.6 Файл cl_application.hpp

Листинг 6 – cl_application.hpp

```
    #ifndef __CL_APPLICATION__HPP
    #define __CL_APPLICATION__HPP
    #include "cl_base.hpp"
    #include "cl_2.hpp"

    class cl_application:public cl_base{
    public:
       cl_application(cl_base* p_head_obj);
       void build_tree_objects();
       int exec_app();
    };

    #endif
```

5.7 Файл cl_base.cpp

Листинг 7 – cl_base.cpp

```
#include "cl_base.hpp"
#include <iostream>
#include <set>
#include <vector>
using namespace std;

cl_base::cl_base(cl_base* p_head_obj, string self_name){
   p_head_object = p_head_obj;
   name = self_name;
   if (p_head_object){
      p_head_object->p_sub_objects.push_back(this);
   }
}

cl_base::~cl_base(){
   for (auto i: p_sub_objects){
      delete i;
   }
}

bool cl_base::set_name(string new_name){
   set<string> copies;
   if (!p_head_object){
      name = new_name;
      return true;
   }
   for (cl_base* obj: p_head_object->p_sub_objects){
      if (copies.count(obj->get_name())){
         return false;
      }
      copies.insert(obj->get_name());
   }
   name = new_name;
   return true;
}

string cl_base::get_name(){
   return name;
}

cl_base* cl_base::get_head_ptr(){
   return p_head_object;
}

cl_base* cl_base::get_sub_object(string obj_name){
   for (cl_base* obj: p_sub_objects){
      if (obj_name == obj->get_name()){
         return obj;
      }
   }
```

```
   return nullptr;
}

void cl_base::print_tree(){
   if (p_sub_objects.size() != 0){
      cout << "\n" << name;
      for (auto i: p_sub_objects){
         cout << "  " << i->get_name();
         i->print_tree();
      }
   }
}
```

5.8 Файл cl_base.hpp

Листинг 8 – cl_base.hpp

```
#ifndef __CL_BASE__HPP
#define __CL_BASE__HPP

#include <string>
#include <vector>
using namespace std;

class cl_base{
private:
   vector<cl_base*> p_sub_objects;
   string name;
   cl_base* p_head_object;
public:
   cl_base(cl_base* p_head_obj, string self_name = "Base_object");
   ~cl_base();
   bool set_name(string new_name);
   string get_name();
   cl_base* get_head_ptr();
   cl_base* get_sub_object(string obj_name);
   void print_tree();
};


#endif
```

5.9 Файл main.cpp

Листинг 9 – main.cpp

```
#include "cl_application.hpp"
#include <iostream>
using namespace std;
/*
УВАЖАЕМЫЙ ГРАЧ ЕВГЕНИЙ ПЕТРОВИЧ!!!
БУДУ ОЧЕНЬ БЛАГОДАРЕН ЕСЛИ В СЛУЧАЕ ВОЗВРАЩЕНИЯ РАБОТЫ ВЫ УТОЧНИТЕ ЧТО КОНКРЕТНО С НЕЙ НЕ ТАК,
ВМЕСТО ТОГО ЧТОБЫ ПИСАТЬ НЕКОНСТРУКТИВНЫЕ ОДНОСЛОЖНЫЕ КОММЕНТАРИИ!!!
ЗАРАНЕЕ БЛАГОДАРЮ!!!
*/

int main()
{
   cl_application obj_cl_application(nullptr);
   obj_cl_application.build_tree_objects();
   return obj_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 6.

Таблица 6 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
obj_root obj_root obj_1 obj_root obj_2 obj_2 obj_3 obj_3 obj_3	obj_root obj_root obj_1 obj_2 obj_2 obj_3	obj_root obj_root obj_1 obj_2 obj_2 obj_3
obj_root obj_8 obj_8	obj_root	obj_root
obj obj obj1 obj obj2 obj100 obj103 obj2 obj5 obj5 obj7 obj8 obj8	obj obj obj1 obj2 obj2 obj5 obj5 obj7	obj obj obj1 obj2 obj2 obj5 obj5 obj7

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).