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 построения дерева и запуска приложения;
- cin/cout - ввод/вывод, if - условный оператор, for - цикл со счетчиком, while - цикл с условием.

Класс cl_base:

- свойства/поля:
  - поле для наименования 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 текущему объекту в дереве иерархии:
    - наименование — subordinate_objects;
    - тип — vector <cl_base *>;
    - модификатор доступа — private;
- функционал:
  - метод cl_base — параметризированный конструктор класса;
  - метод ~cl_base — деструктор класса;
  - метод set_name — метод редактирования имени объекта;
  - метод get_name — метод получения имени объекта;

- метод get_subordinate_object — метод получения указателя на непосредственно подчиненный объект по имени;
- метод print_tree — метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- метод get_p_head — метод получения указателя на головной объект текущего объекта.

Класс cl_1:

- функционал:
  - метод cl_1 — параметризированный конструктор класса.

Класс cl_application:

- функционал:
  - метод cl_application — параметризированный конструктор класса;
  - 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_1	public		2
		cl_application	public		3
2	cl_1			класс наследуемый от класса cl_base	
3	cl_application			класс объекта приложения	

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 класса.

Параметры: cl_base * p_head_object - указатель на головной объект, string s_object_name - имя узла дерева, по умолчанию подается "Base object".

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		полю p_head_object этого объекта присваивается параметр p_head_object	2
2		полю s_object_name этого объекта присваивается параметр s_object_name	3
3	p_head_object ненулевой	в вектор указателей подчиненных объектов головного объекта добавляется указатель на этот объект	∅
			∅

3.2 Алгоритм деструктора класса cl_base

Функционал: деструктор класса.

Параметры: нет.

Алгоритм деструктора представлен в таблице 3.

Таблица 3 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация i = 0	2
2	i < длины subordinate_objects	удаляем объект по i-ому указателю вектора указателей subordinate_objects	3
			3
3		i++	∅

3.3 Алгоритм метода set_name класса cl_base

Функционал: метод редактирования имени объекта.

Параметры: string new_name - новое наименование объекта.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_name класса cl_base

№	Предикат	Действия	№ перехода
1	p_head_object этого объекта ненулевой	инициализация i = 0	2
			5
2	i < длины subordinate_objects объекта по указателю p_head_object		3
			5
3	имя объекта по i-ому указателю вектора указателей subordinate_objects объекта по указателю p_head_object == new_name	возвращаем ложь	∅

№	Предикат	Действия	№ перехода
			4
4		i++	5
5		полю s_object_name этого объекта присваивается new_name	6
6		возвращаем истину	∅

3.4 Алгоритм метода get_name класса cl_base

Функционал: метод получения имени объекта.

Параметры: нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода
1		возвращаем s_object_name	∅

3.5 Алгоритм метода get_subordinate_object класса cl_base

Функционал: метод получения указателя на непосредственно подчиненный объект по имени.

Параметры: string search_name - поле искомого подчиненного объекта.

Возвращаемое значение: cl_base* - указатель на непосредственно подчиненный объект по его имени.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_subordinate_object класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация i= 0	2
2	i < длины subordinate_objects		3
			5
3	поле s_object_name объекта i-ому указателю вектора указателей subordinate_objects == search_name	возвращаем объект по i-ому указателю вектора указателей subordinate_objects	∅
			4
4		i++	5
5		возвращаем нулевой указатель	∅

3.6 Алгоритм конструктора класса cl_1

Функционал: параметризированный конструктор класса.

Параметры: cl_base* p_head_object - указатель на объект базового класса, содержащий указатель головного объекта в дереве иерархии, string s_object_name - наименование создаваемого объекта.

Алгоритм конструктора представлен в таблице 7.

Таблица 7 – Алгоритм конструктора класса cl_1

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head_object и s_object_name	∅

3.7 Алгоритм конструктора класса cl_application

Функционал: параметризированный конструктор класса.

Параметры: cl_base* p_head_object - указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметром p_head_object	∅

3.8 Алгоритм метода build_tree_objects класса cl_application

Функционал: метод построения исходного дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		объявление строк s_head_name и s_sub_name	2
2		объявление указателя p_head на объекта класса cl_base	3
3		инициализируем указатель p_sub на объект класса cl_base нулевым	4
4		ввод значения s_head	5
5		вызов метода set_name этого объекта с параметром s_head_name	6
6		присваиваем объекту по указателю p_head этот объект	7
7		ввод значения s_head_name и s_sub_name	8

№	Предикат	Действия	№ перехода
8	s_head_name равно s_sub_name		∅
			9
9	p_sub ненулевой и s_head равно возвращаемому значению метода get_name объекта по указателю p_sub	присваиваем p_head значение p_sub	10
			10
10	возвращаемое значение метода get_subordinate_object с параметром s_sub объекта по указателю p_head - нулевой, и s_head_name равно возвращаемому значению метода get_name объекта по указателю p_head	присваиваем объекту по указателю p_sub новый объект класса cl_1 с параметрами p_head и s_sub_name	7
			7

3.9 Алгоритм метода exec_app класса cl_application

Функционал: метод запуска приложения.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		вывод значения возвращаемого значения метода get_name этого объекта	2

№	Предикат	Действия	№ перехода
2		вызов метода print_tree этого объекта	3
3		возвращаем 0	∅

3.10 Алгоритм метода print_tree класса cl_base

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

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1	длина subordinate_objects не равна 0	вывод перенос строки и возвращаемого значения метода get_name этого объекта	2
			∅
2		инициализация i = 0	3
3	i < длины subordinate_objects	вывод " " и возвращаемого значения метода get_name объекта по i-ому указателю вектора указателей subordinate_objects	4
			∅
4		вызов print_tree для объекта по i-ому указателю вектора указателей subordinate_objects	5
5		i++	∅

3.11 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 12.

Таблица 12 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создаем объект ob_cl_application класса cl_application в параметре подается нулевой указатель	2
2		вызываем метод build_tree_objects объекта ob_cl_application	3
3		вызываем метод exec_app объекта ob_cl_application	∅

3.12 Алгоритм метода get_p_head класса cl_base

Функционал: метод получения указателя на головной объект текущего объекта.

Параметры: нет.

Возвращаемое значение: cl_base* - указатель на головной объект текущего объекта.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода get_p_head класса cl_base

№	Предикат	Действия	№ перехода
1		возвращаем указатель на головной объект текущего объекта p_head_object	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

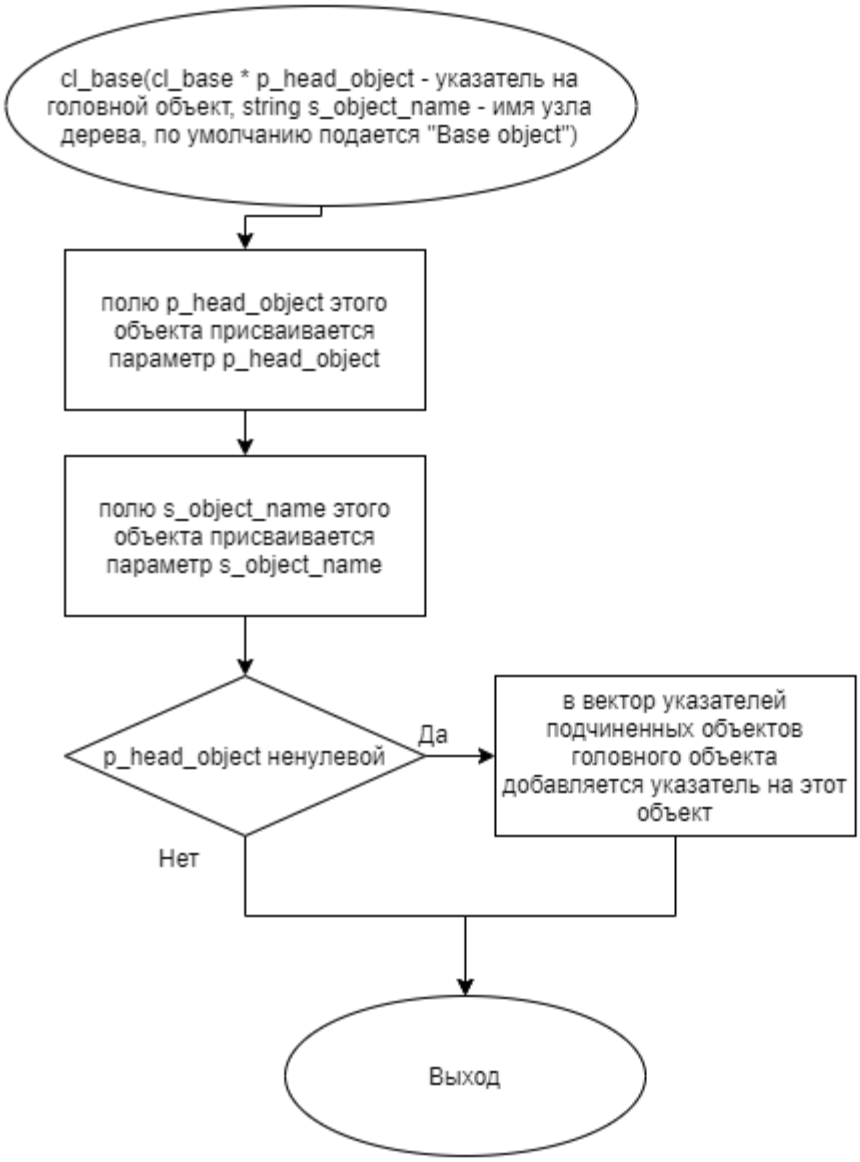

Рисунок 1 – Блок-схема алгоритма

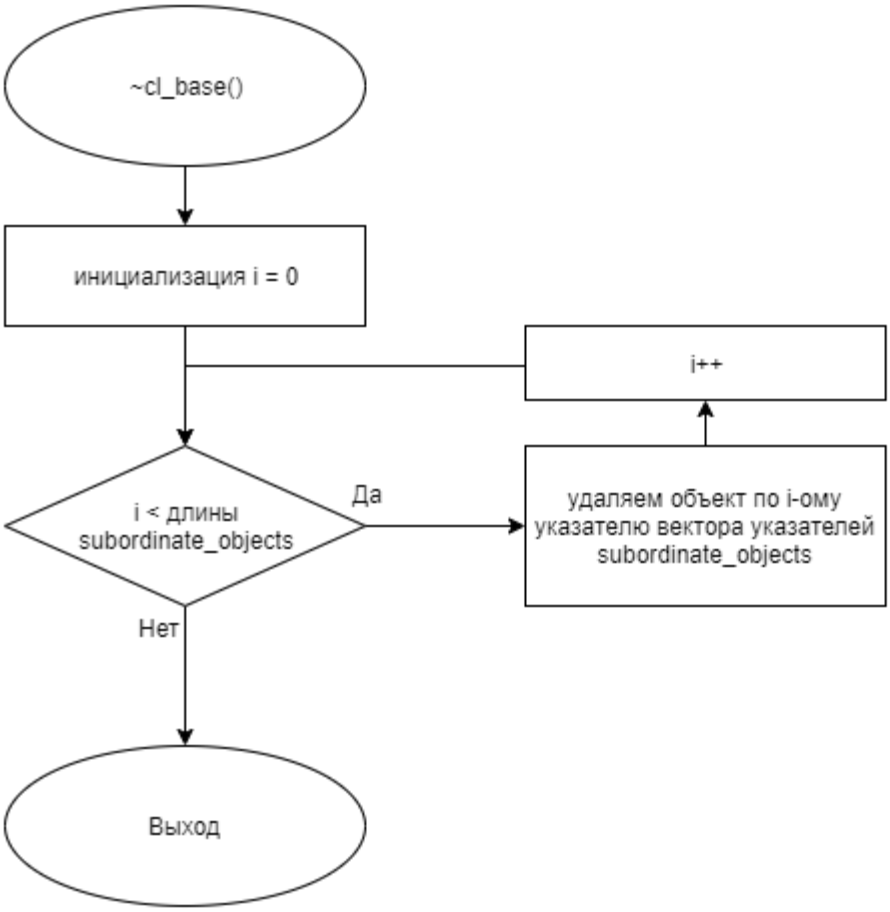

Рисунок 2 – Блок-схема алгоритма

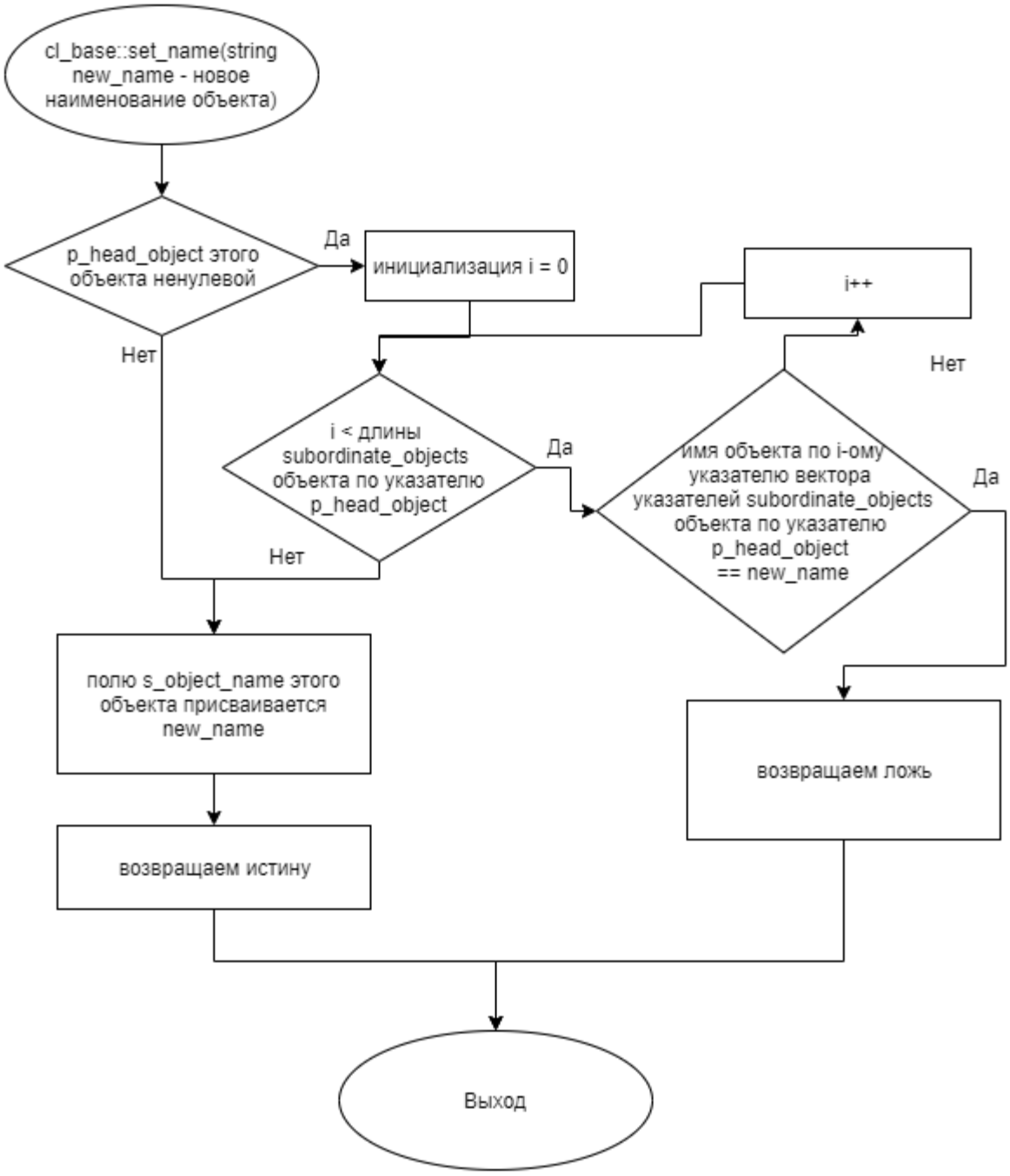

Рисунок 3 – Блок-схема алгоритма

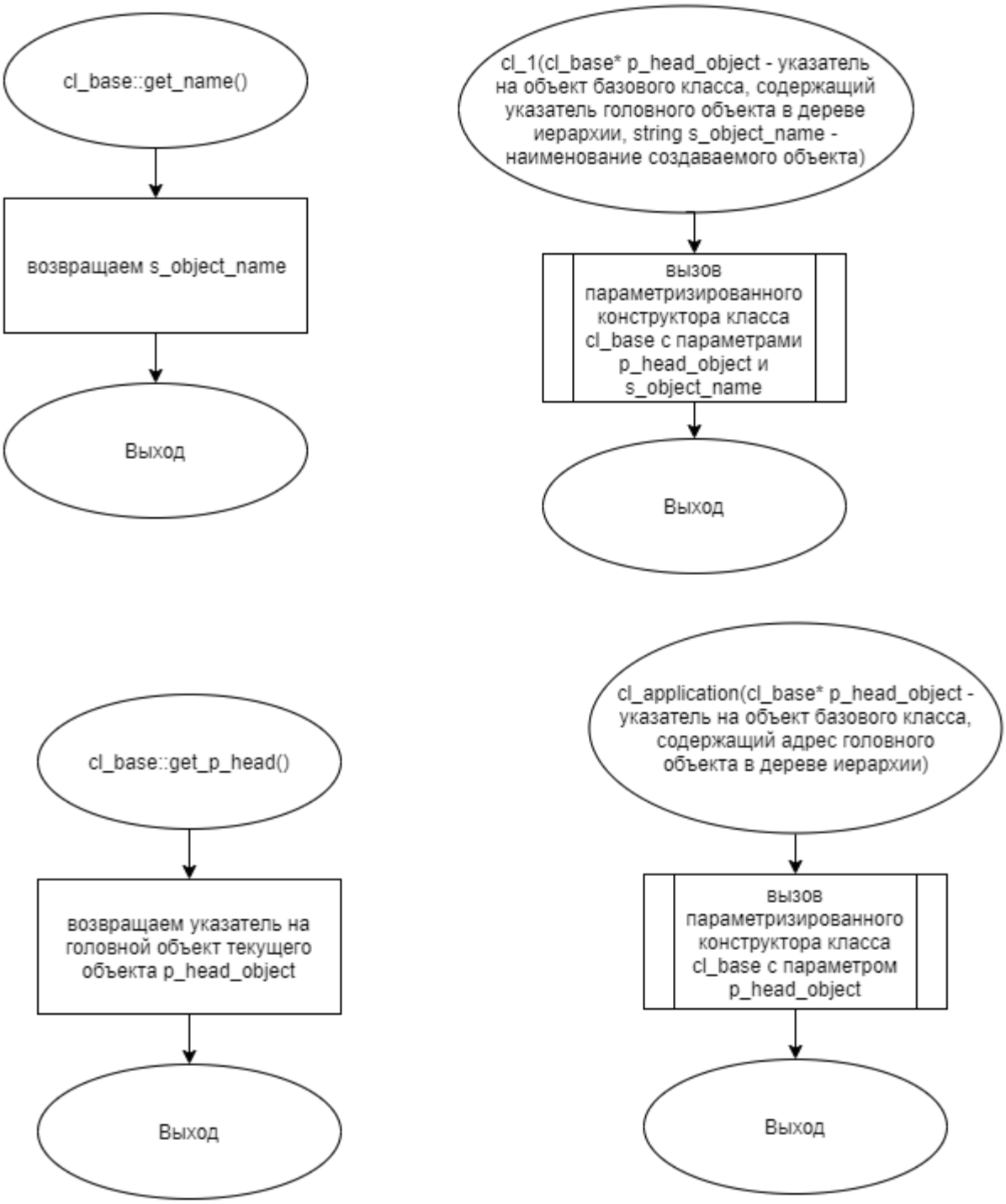

Рисунок 4 – Блок-схема алгоритма

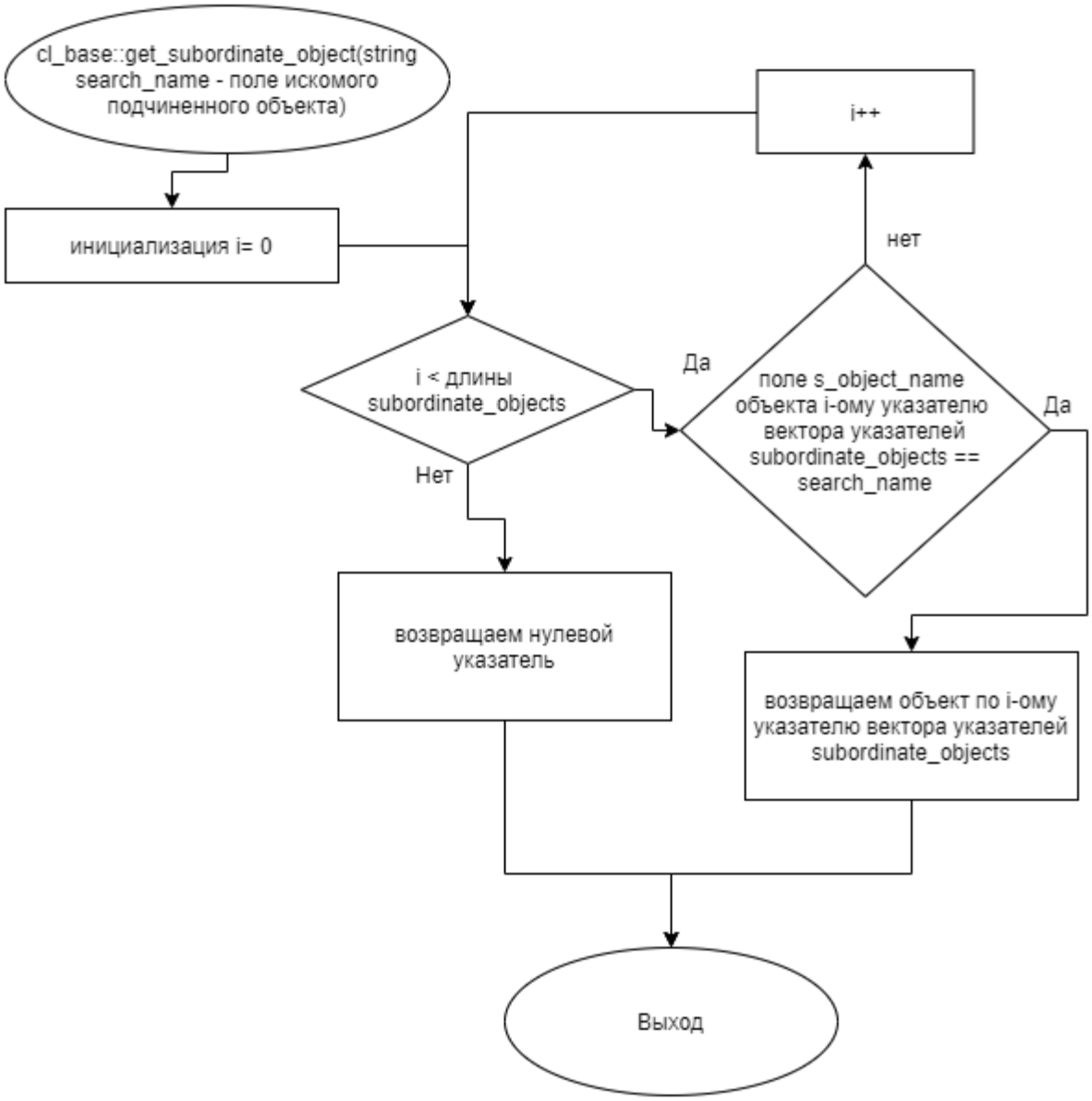

Рисунок 5 – Блок-схема алгоритма

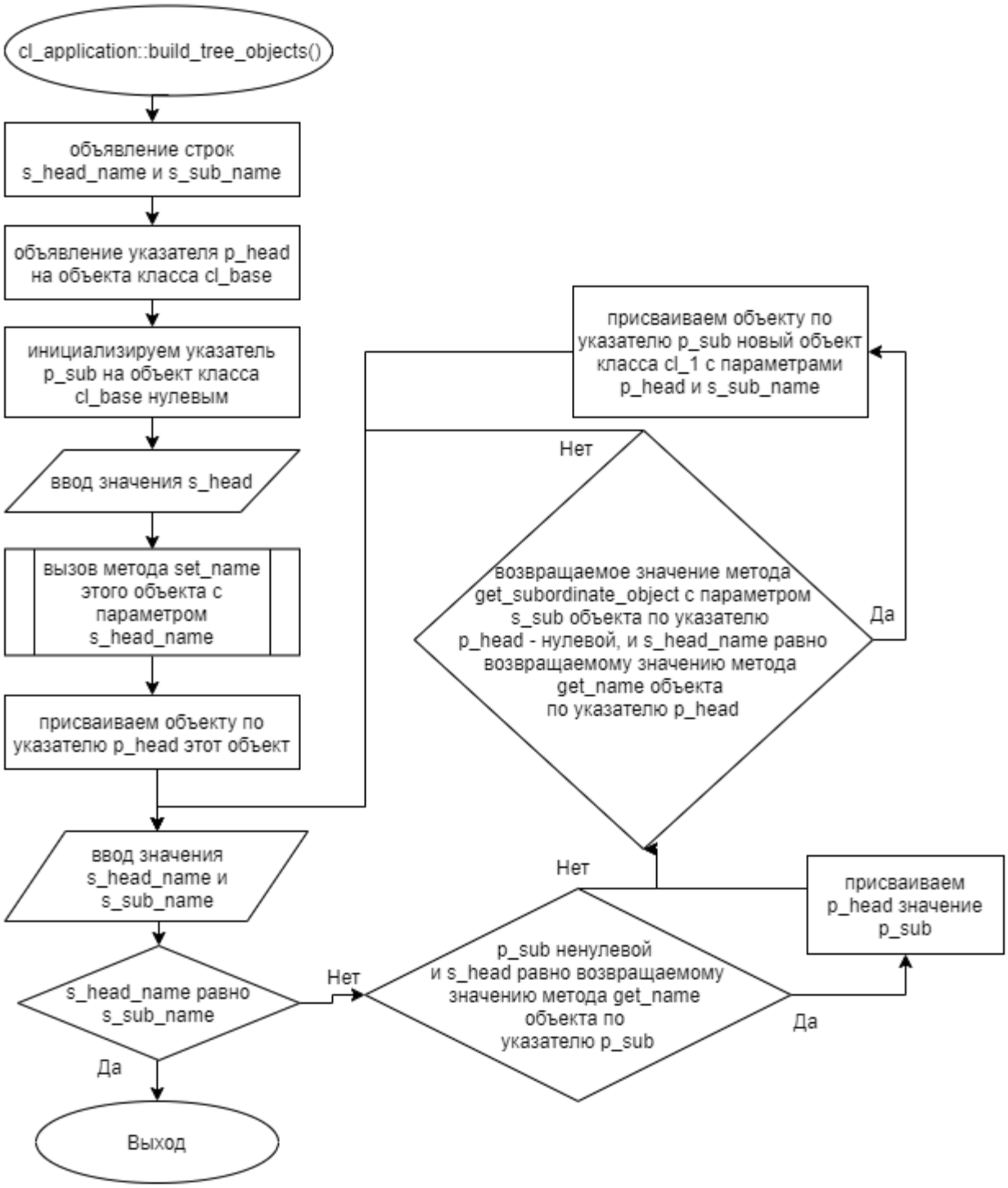

Рисунок 6 – Блок-схема алгоритма

Рисунок 7 – Блок-схема алгоритма

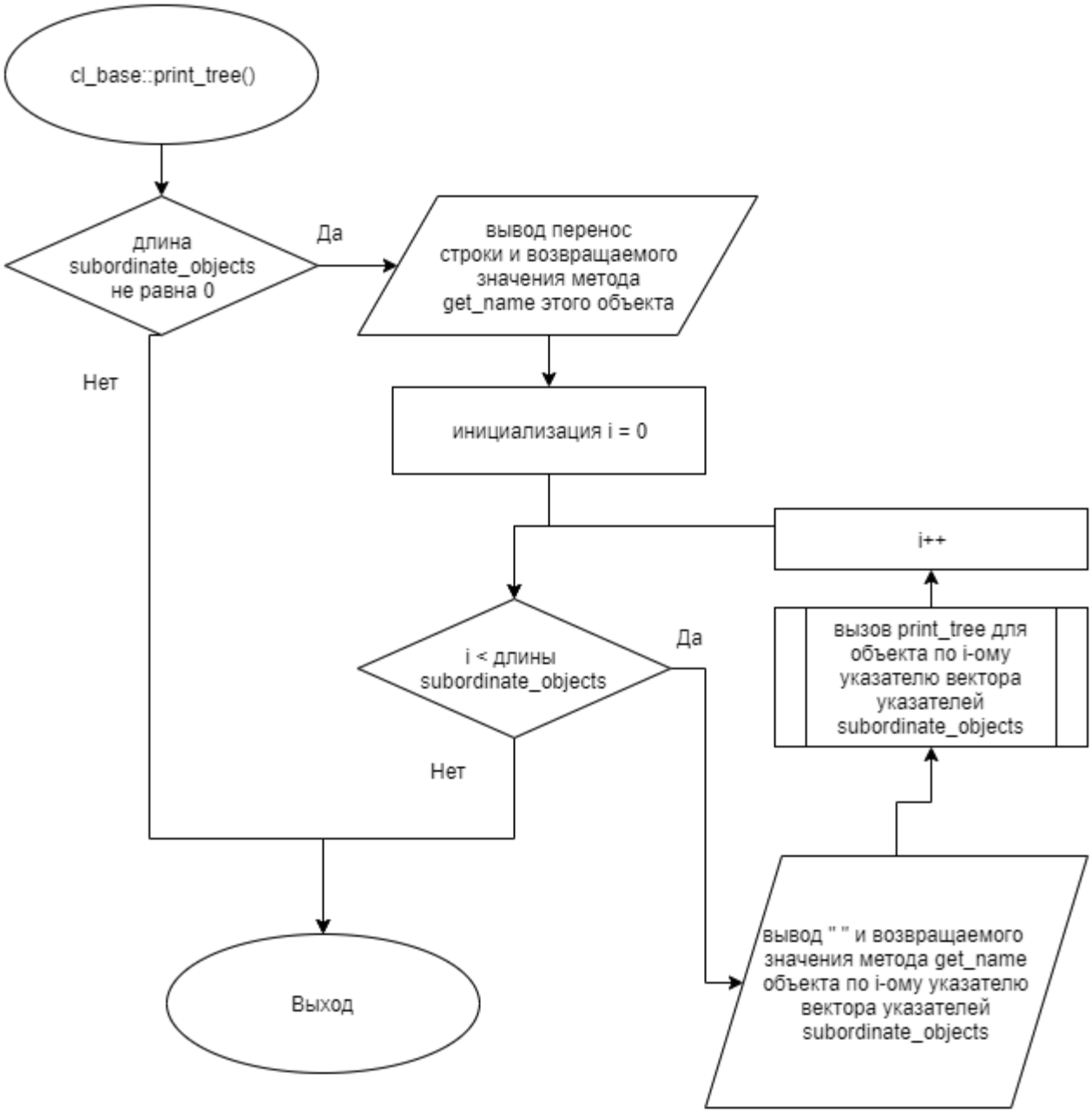

Рисунок 8 – Блок-схема алгоритма

Рисунок 9 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"

// вызов параметризированного конструктора класса cl_base с параметрами p_head_object и s_object_name
cl_1::cl_1(cl_base*      p_head_object,      string      s_object_name)      : cl_base(p_head_object, s_object_name){}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef __CL_1__H
#define __CL_1__H

#include "cl_base.h"


class cl_1: public cl_base{
public:
   // конструктор, создающий объект класса cl_1
   cl_1(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.3 Файл cl_application.cpp

Листинг 3 – cl_application.cpp

```
#include "cl_application.h"
```

```
// вызов параметризированного конструктора класса cl_base с параметром p_head_object
cl_application::cl_application(cl_base * p_head_object) : cl_base(p_head_object){}


void cl_application::build_tree_objects(){
   /*
   метод построения дерева иерархии объектов
   */

   string s_head_name, s_sub_name;
   cl_base* p_head;
   cl_base* p_sub = nullptr;

   cin >> s_head_name;
   // вызов метода set_name этого объекта с параметром s_head_name
   this -> set_name(s_head_name);
   // присваиваем p_head этот объект
   p_head = this;

   while(true){
      cin >> s_head_name >> s_sub_name;
      // проверка на схожесть объекта дерева иерархии (родительского и производного)
      if (s_head_name == s_sub_name){
         break;
      }
      // если p_sub ненулевой и s_head_name равен возвращаемому значению метода get_name объекта по указателю p_sub
      if (p_sub != nullptr && s_head_name == p_sub -> get_name()){
         p_head = p_sub;
      }
      //возвращаемое значение метода get_subordinate_object с параметром s_sub объекта по указателю p_head - нулевой,
      // и s_head_name равно возвращаемому значению метода get_name объекта по указателю p_head
      if (p_head -> get_subordinate_object(s_sub_name) == nullptr && s_head_name == p_head -> get_name()){

         // присваиваем объекту по указателю p_sub новый объект класса cl_1 с параметрами p_head и s_sub_name
         p_sub = new cl_1(p_head, s_sub_name);
      }
   }
}

int cl_application::exec_app(){
   // вывод значения возвращаемого значения метода get_name этого объекта
   cout << this -> get_name();

   //вызов метода print_tree этого объекта
   this -> print_tree();
```

```
   return 0;
}
```

5.4 Файл cl_application.h

Листинг 4 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H

#include "cl_base.h"
#include "cl_1.h"

class cl_application : public cl_base{
public:
   cl_application(cl_base * p_head_object); // конструктор, создающий объект
класса cl_app..
   void build_tree_objects(); // метод, создающий иерархию объекта
   int exec_app(); // метод запуска приложения
};

#endif
```

5.5 Файл cl_base.cpp

Листинг 5 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base * p_head_object, string s_object_name){
   /*
   параметризированный конструктор
   p_head_object - указатель на головной объект
   s_object_name - имя узла дерева
   */

   //полю p_head_object этого объекта присваивается параметр p_head_object
   //полю s_object_name этого объекта присваивается параметр s_object_name
   this -> p_head_object = p_head_object;
   this -> s_object_name = s_object_name;

   // если p_head_object ненулевой, то в subordinate_objects добавляется
указатель на этот объект
   if (p_head_object){
      p_head_object -> subordinate_objects.push_back(this);
```

```
   }
}
cl_base::~cl_base(){
   // проходимся по каждому элементу subordinate_objects и удаляем его
   for   (int i =0; i < subordinate_objects.size(); i++){
      delete subordinate_objects[i];
   }
}

bool cl_base::set_name(string new_name){
   /*
   метод редактирования имени объекта
   new_name - новое имя узла дерева
   */

   // проходимся по i-ому указателю вектора указателей subordinate_objects
объекта по указателю p_head_object
   // если он равен new_name, то возвращаем false
   if (p_head_object != nullptr){
      for (int i = 0; i < p_head_object -> subordinate_objects.size(); i++){
         if  (p_head_object  ->  subordinate_objects[i]  ->  get_name()  ==
new_name){
            return false;
         }
      }
   }
   // полю s_object_name этого объекта присваивается new_name
   this -> s_object_name = new_name;
   return true;
}

string cl_base::get_name(){
   // возвращаем s_object_name
   return s_object_name;
}

cl_base * cl_base::get_p_head(){
   // возвращаем p_head_object
   return p_head_object;
}

cl_base * cl_base::get_subordinate_object(string search_name){
   /*
   получение указателя на непосредственно подчиненный объект по имени
   search_name - имя искомого объекта
   */

   // проходимся по i-ому указателю вектора указателей subordinate_objects,
если он равен search_name
   // возвращаем i-ый subordinate_objects
   for (int i = 0; i < subordinate_objects.size(); i++){
      if (subordinate_objects[i] -> get_name() == search_name){
         return subordinate_objects[i];
      }
   }
```

```
   // иначе возвращается нулевой указатель
   return nullptr;
}

void cl_base::print_tree(){
   if (subordinate_objects.size() != 0){
      // выводим дерево объекта
      cout << endl << this -> get_name();

      // проходимся по subordinate_objects
      for (int i = 0; i < subordinate_objects.size(); i++){
         // выводим объект i-ого subordinate_objects
         cout << "  " << subordinate_objects[i] -> get_name();

         // создается рекурсия, если у i-ого subordinate_objects есть еще
какие-то привязанные объекты
         subordinate_objects[i] -> print_tree();
      }
   }
}
```

5.6 Файл cl_base.h

Листинг 6 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H

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

using namespace std;

class cl_base {
private:
   string s_object_name; // имя объекта
   cl_base * p_head_object; // указатель на родителя
   vector <cl_base *> subordinate_objects; // вектор детей объекта
public:
   cl_base(cl_base * p_head_object, string s_object_name = "Base object"); //
параметризированный конструктор
   string get_name();
   cl_base * get_p_head();
   bool set_name(string new_name); //метод редактирования имени объекта
   void print_tree(); // метод вывода дерева иерархии
   cl_base   *   get_subordinate_object(string   search_name);   //   получение
указателя на непосредственно подчиненный объект по имени
   ~cl_base();
};
```

```
#endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

```
#include "cl_application.h"

int main()
{
   cl_application ob_cl_application(nullptr);
   ob_cl_application.build_tree_objects();

   return ob_cl_application.exec_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_2 Object_3 Object_2 Object_4 Object_5 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_2 Object_2 Object_3 Object_4	Object_root Object_root Object_1 Object_2 Object_2 Object_3 Object_4
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_root Object_4 Object_2 Object_3 Object_2 Object_4 Object_4 Object_5 Object_5 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_4 Object_4 Object_5	Object_root Object_root Object_1 Object_2 Object_3 Object_4 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).