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_app класса cl_app предназначен для запуск программы;
- cin - объект стандартного потока ввода;
- cout - объект стандартного потока вывода.

Класс cl_base:

- свойства/поля:
    - поле строковое поле:
        - наименование — s_name;
        - тип — string;
        - модификатор доступа — private;
    - поле указатель на объект:
        - наименование — p_head_object;
        - тип — string;
        - модификатор доступа — private;
    - поле вектор указателей на объект:
        - наименование — p_sub_object;
        - тип — string;
        - модификатор доступа — private;
- функционал:
    - метод Конструктор cl_base — параметризированный конструктор;
    - метод Метод set_name — установка имени;
    - метод Метод get_name — возвращение имени;
    - метод Метод get_head — возвращение указателя объекта-родителя;
    - метод Метод print_tree — вывод названий всех объектов;
    - метод Метод get_sub_obj — поиск дочернего объекта по имени и

возвращение указателя на него;

- метод Деструктор ~cl_base — отчистка памяти памяти под объекты.

Класс cl_app:

- функционал:
  - метод Конструктор cl_app — параметризированный конструктор;
  - метод Метод build_tree_objects — вывод дерева объекта;
  - метод Метод exec_app — запуск программы.

Класс cl_1:

- функционал:
  - метод Конструктор cl_base — параметризированный конструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base				
		cl_app	public		2
		cl_1	public		3
2	cl_app				
3	cl_1				

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: cl_base *p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		присваивание полю s_name класса cl_base значения s_name параметра конструктора	2
2		присваивание полю p_head_object класса cl_base значения p_head_object параметра конструктора	3
3	присутствует ли указатель на объект выше по иерархии	добавление указателя на этот объект в вектор p_sub_objects	∅
			∅

3.2 Алгоритм метода set_name класса cl_base

Функционал: проверка на совпадение имен.

Параметры: string s_new_name.

Возвращаемое значение: true или false (bool).

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода set_name класса cl_base

№	Предикат	Действия	№ перехода
1	присутствует ли указатель на объект выше по иерархии		2
			3
2	проверяем занято ли имя	возврат true	∅
			3
3		устанавливается новое имя для текущего объекта	4
4		возврат true	∅

3.3 Алгоритм метода get_name класса cl_base

Функционал: возвращение имени.

Параметры: none.

Возвращаемое значение: get_name.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода
1		возврат s_name	∅

3.4 Алгоритм метода get_head класса cl_base

Функционал: возвращение указателя объекта-родителя.

Параметры: none.

Возвращаемое значение: get_name.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_head класса cl_base

№	Предикат	Действия	№ перехода
1		возврат p_head_object	∅

3.5 Алгоритм метода print_tree класса cl_base

Функционал: вывод дерева.

Параметры: none.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1	объекты ниже по иерархии отсутсвуют	return	∅
			2
2		вывод имени этого объекта	3
3		вызов метода print_tree() для каждого объекта ниже по иерархии	∅

3.6 Алгоритм метода get_sub_obj класса cl_base

Функционал: вывод i объекта.

Параметры: string s_name.

Возвращаемое значение: p_sub_objects.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_sub_obj класса cl_base

№	Предикат	Действия	№ перехода
1		переменной i типа int присваевается значение 0	2

№	Предикат	Действия	№ перехода
2	i<p_sub_object		3
			4
3	p_sub_objects[i]->s_name==s_name	возврат p_sub_objects[i], i++	2
		i++	2
4		возврат значения 0	∅

3.7 Алгоритм деструктора класса cl_base

Функционал: отчистка памяти памяти под объекты.

Параметры: none.

Алгоритм деструктора представлен в таблице 8.

Таблица 8 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		отчистка памяти под каждый из объектов в векторе p_sub_objects	∅

3.8 Алгоритм конструктора класса cl_app

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_object.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_app

№	Предикат	Действия	№ перехода
1		присваивание полю s_name класса cl_base значения s_name параметра	2
2		присваивание полю p_head_object класса cl_base значения p_head_object параметра конструктора	3

№	Предикат	Действия	№ перехода
3	присутствует ли указатель на объект выше по иерархии	добавление указателя на этот объект в вектор p_sub_objects	∅
			∅

3.9 Алгоритм метода build_tree_objects класса cl_app

Функционал: построение дерева объекта.

Параметры: none.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода build_tree_objects класса cl_app

№	Предикат	Действия	№ перехода
1		объявление переменных s_head, s_sub типа string	2
2		инициализация указателей p_head, p_sub на объекты типа cl_base со значением nullptr	3
3		ввод значения s_head	4
4		вызов метода set_name(s_head)	5
5		начало бесконечного цикла	6
6		ввод значений s_head и s_sub	7
7	s_head==s_sub	выход из цикла	∅
			8
8	p_sub!=nullptr и s_head==имени p_sub	p_head=p_sub	9
			9
9	объекта ниже по иерархии не существует	создание указателя на объект типа cl_1(p_head, s_sub)	5
			5

3.10 Алгоритм метода exec_app класса cl_app

Функционал: запуск программы.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода exec_app класса cl_app

№	Предикат	Действия	№ перехода
1		вывод значения метода get_name()	2
2		вызов метода print_tree()	∅

3.11 Алгоритм конструктора класса cl_1

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_1

№	Предикат	Действия	№ перехода
1		присваивание полю s_name класса cl_base значения s_name параметра конструктора	2
2		присваивание полю p_head_object класса cl_base значения p_head_object параметра конструктора	3
3	присутствует ли указатель на объект выше по иерархии	добавление указателя на этот объект в вектор p_sub_objects	∅
			∅

3.12 Алгоритм функции main

Функционал: основная функция программы.

Параметры: none.

Возвращаемое значение: string.

Алгоритм функции представлен в таблице 13.

Таблица 13 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_cl_app класса cl_app с параметром nullptr	2
2		вызов метода build_tree_objects() для этого объекта	3
3		возврат значений метода exec_app()	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-8.

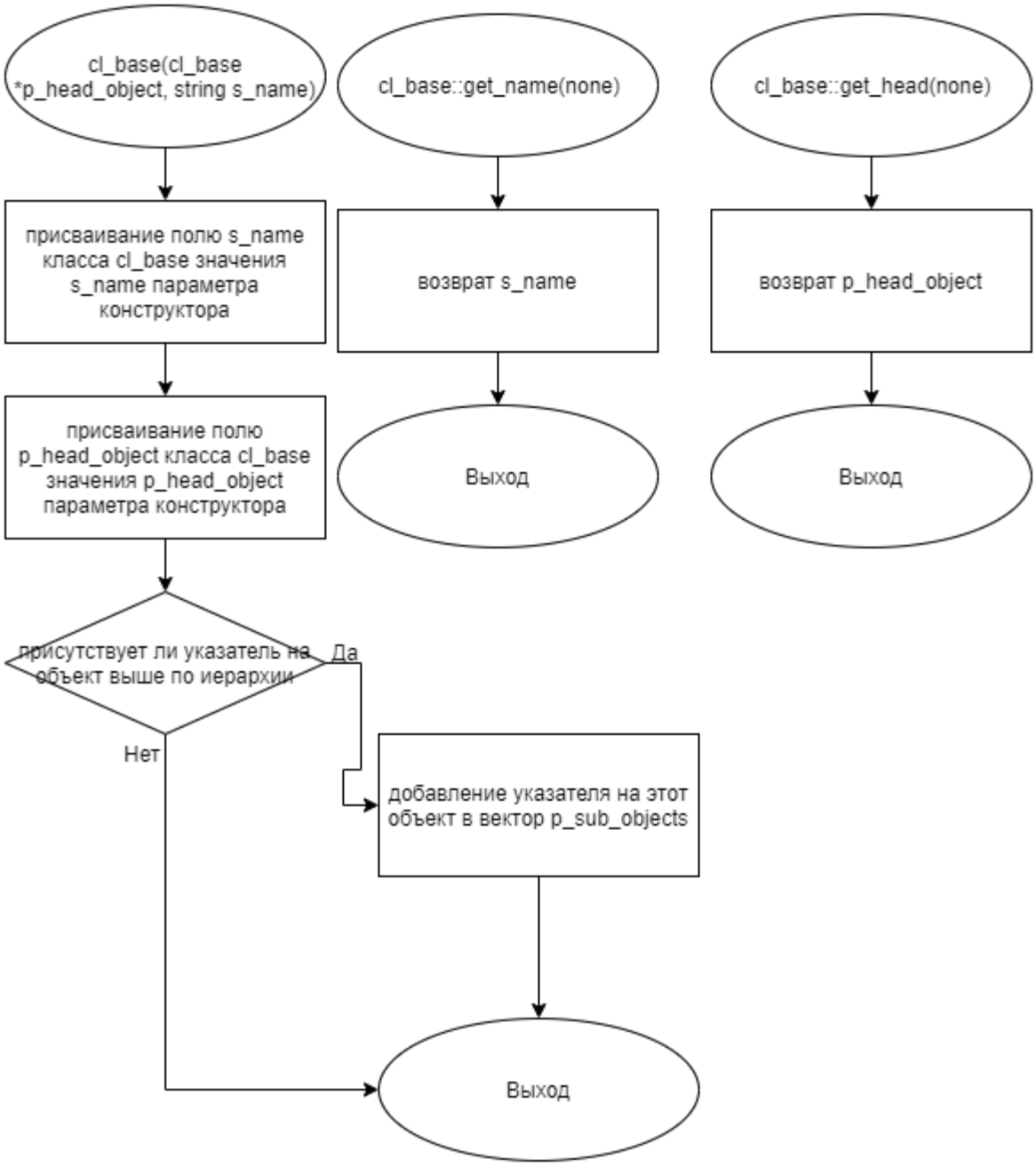

Рисунок 1 – Блок-схема алгоритма

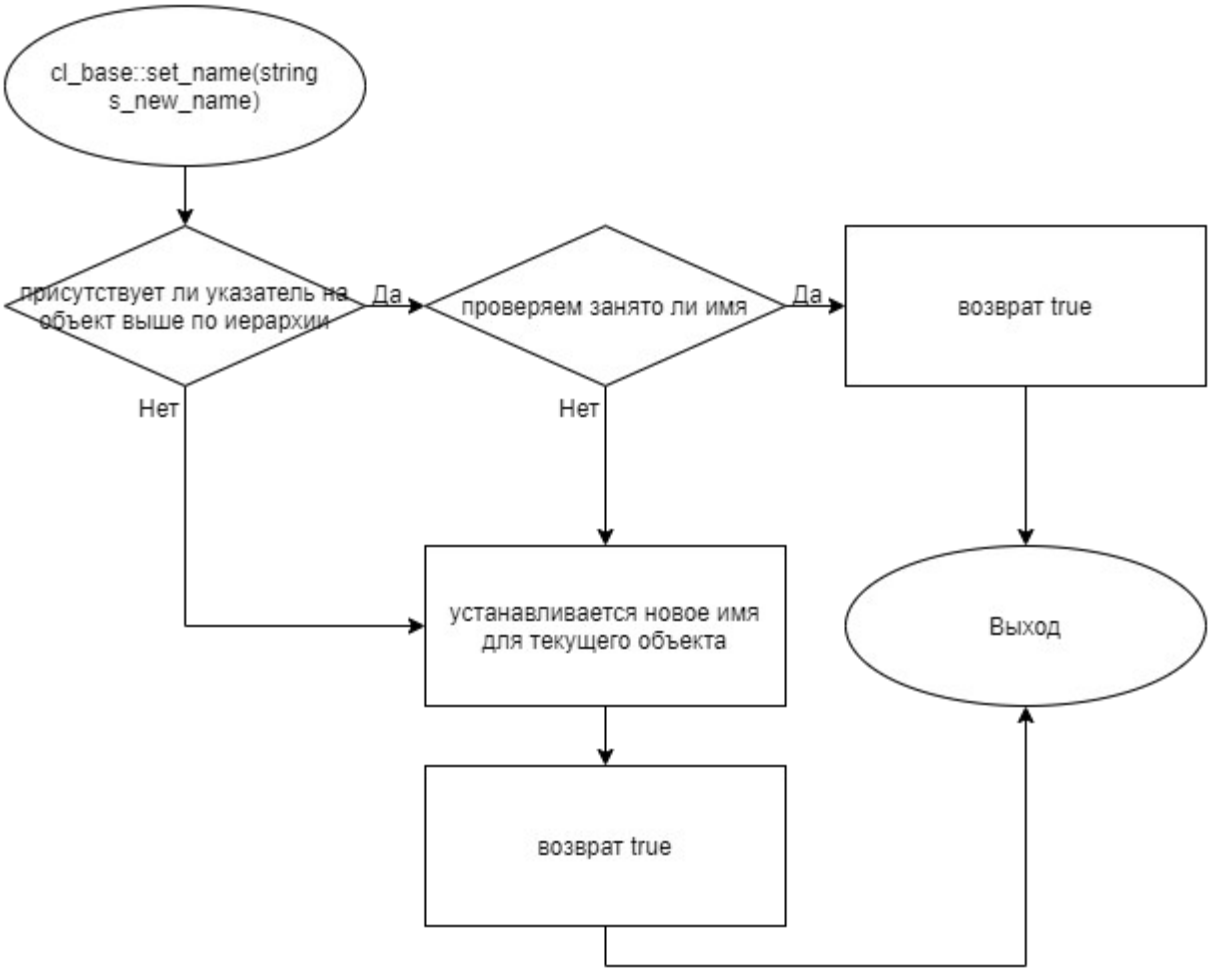

Рисунок 2 – Блок-схема алгоритма

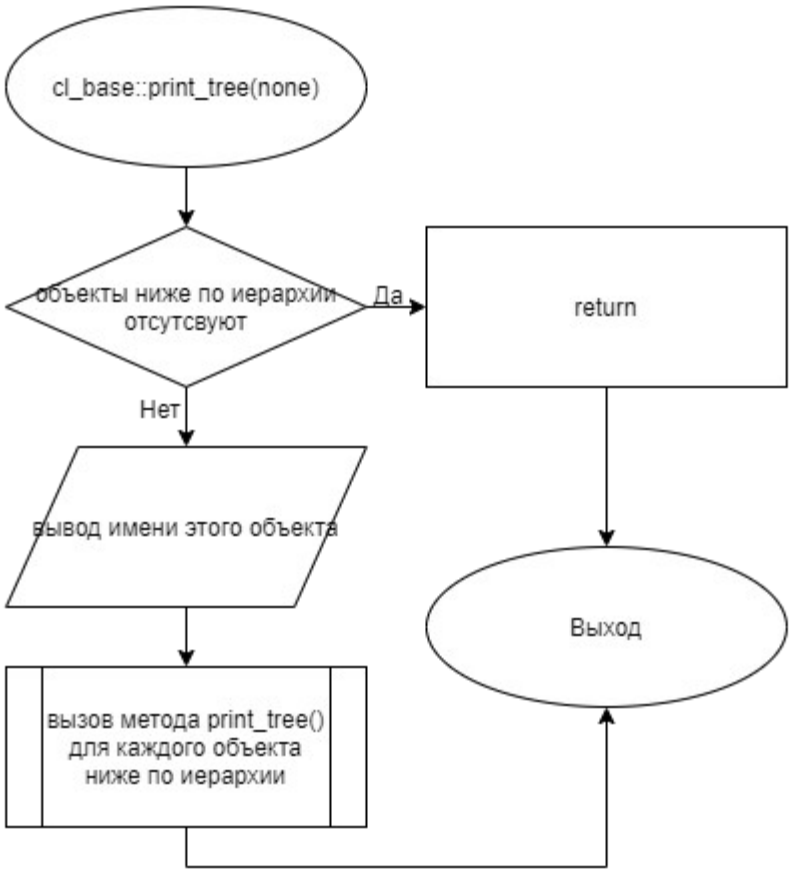

Рисунок 3 – Блок-схема алгоритма

Рисунок 4 – Блок-схема алгоритма

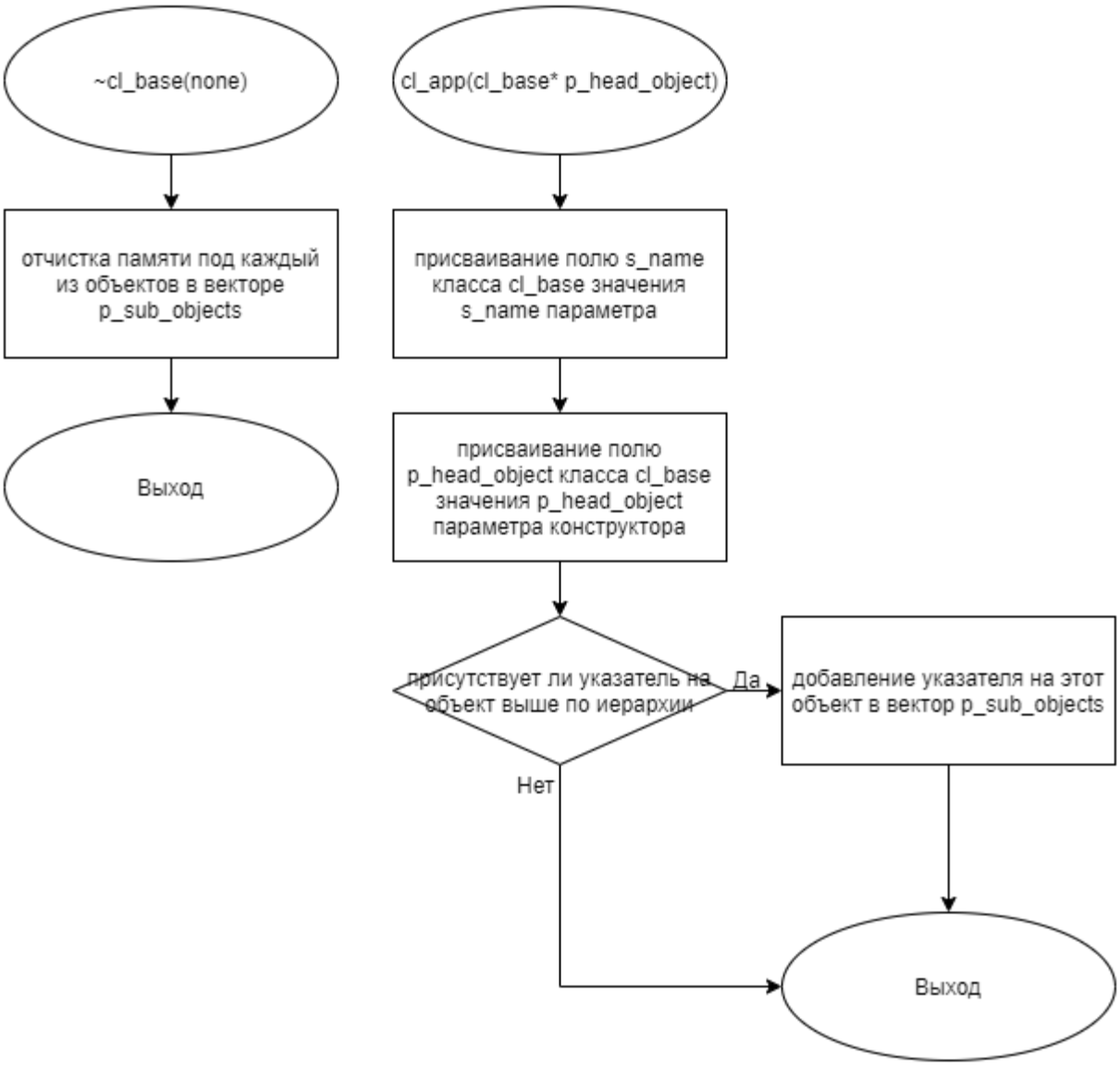

Рисунок 5 – Блок-схема алгоритма

Рисунок 6 – Блок-схема алгоритма

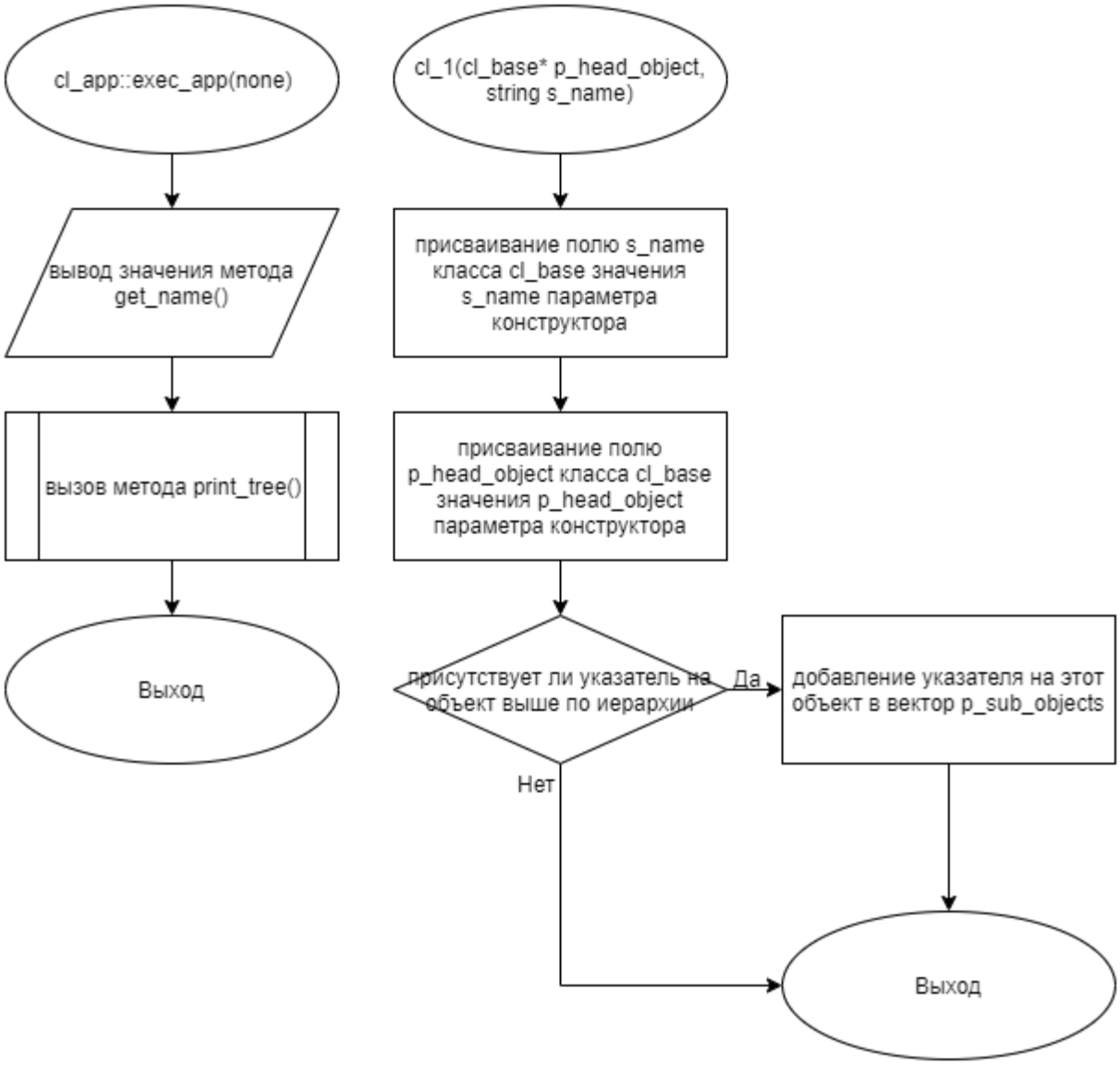

Рисунок 7 – Блок-схема алгоритма

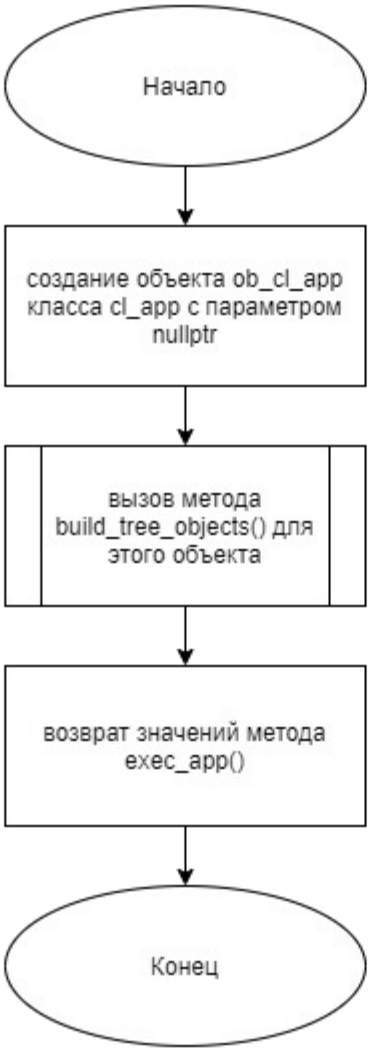

Рисунок 8 – Блок-схема алгоритма

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"

cl_1::cl_1(cl_base*   p_head_object,   string   s_name):cl_base(p_head_object,
s_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:
      cl_1(cl_base* p_head_object, string s_name);
};

#endif
```

5.3 Файл cl_app.cpp

Листинг 3 – cl_app.cpp

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

using namespace std;
```

```
cl_app::cl_app(cl_base* p_head_object):cl_base(p_head_object){}
void cl_app::build_tree_objects()
{
   string s_head, s_sub;
   cl_base* p_head=this, *p_sub=nullptr;
   cin>>s_head;
   set_name(s_head);
   while (true)
   {
      cin>>s_head>>s_sub;
      if (s_head==s_sub)
      {
         break;
      }
      if (p_sub!=nullptr&&s_head==p_sub->get_name())
      {
         p_head=p_sub;
      }
      if (p_head->get_sub_obj(s_sub)==nullptr && s_head==p_head->get_name())
      {
         p_sub=new cl_1(p_head, s_sub);
      }
   }
}

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

5.4 Файл cl_app.h

Листинг 4 – cl_app.h

```
#ifndef __CL_APP__H
#define __CL_APP__H
#include "cl_base.h"
#include "cl_1.h"

class cl_app: public cl_base
{
   public:
      cl_app(cl_base* p_head_object);
      void build_tree_objects();
      int exec_app();
};
```

```
#endif
```

5.5 Файл cl_base.cpp

Листинг 5 – cl_base.cpp

```
#include "cl_base.h"
#include <iostream>
#include <vector>

using namespace std;

cl_base::cl_base(cl_base* p_head_object, string s_name)
{
   this->s_name=s_name;
   this->p_head_object=p_head_object;
   if (p_head_object!=nullptr)
   {
      p_head_object->p_sub_objects.push_back(this);
   }
}
cl_base::~cl_base()
{
   for (int i=0; i<p_sub_objects.size(); i++)
   {
      delete p_sub_objects[i];
   }
}
bool cl_base::set_name(string s_new_name)
{
   if (get_head()!=nullptr)
   {
      for (int i=0; i<get_head()->p_sub_objects.size(); i++)
      {
         if (get_head()->p_sub_objects[i]->get_name()==s_new_name)
         {
            return false;
         }
      }
   }
   s_name=s_new_name;
   return true;
}
void cl_base::print_tree()
{
   if (p_sub_objects.size()==0)
   {
      return;
   }
   else
```

```
        {
           cout<<endl<<get_name();
           for (int i=0; i<p_sub_objects.size(); i++)
           {
              cout<<"  "<<p_sub_objects[i]->get_name();
           }
           for (int i=0; i<p_sub_objects.size(); i++)
           {
              p_sub_objects[i]->print_tree();
           }
        }
     }
     string cl_base::get_name()
     {
        return s_name;
     }
     cl_base* cl_base::get_head()
     {
        return p_head_object;
     }
     cl_base* cl_base::get_sub_obj(string s_name)
     {
        for (int i=0; i<p_sub_objects.size(); i++)
        {
           if (p_sub_objects[i]->s_name==s_name)
           {
              return p_sub_objects[i];
           }
        }
        return 0;
     }
```

5.6 Файл cl_base.h

Листинг 6 – cl_base.h

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

     using namespace std;

     class cl_base
     {
        public:
           cl_base(cl_base* p_head_object, string s_name="Base Object");
           bool set_name(string s_new_name);
           string get_name();
           cl_base* get_head();
```

```
      void print_tree();
      cl_base* get_sub_obj(string s_name);
      ~cl_base();
   private:
      string s_name;
      cl_base* p_head_object;
      vector <cl_base*> p_sub_objects;
};
#endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

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

int main()
{
   cl_app ob_cl_app(nullptr);
   ob_cl_app.build_tree_objects();
   return ob_cl_app.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

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).