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 Для построения системы и запуска выполнения;
- cin/cout - Объекты стандартного потока ввода/вывода на экран;
- if..else - Условный оператор;
- for - Итерируемый цикл-счётчик;
- while - Цикл с предусловием.

Класс cl_base:

- свойства/поля:
  - поле Имя объекта:
    - наименование — name;
    - тип — string;
    - модификатор доступа — protected;
  - поле Родительский объект:
    - наименование — header_ptr;
    - тип — *cl_base;
    - модификатор доступа — protected;
  - поле Вектор подчинённых объектов:
    - наименование — children_ptr;
    - тип — vector<cl_base*>;
    - модификатор доступа — protected;
- функционал:
  - метод cl_base — Параметризированный конструктор;
  - метод edit_name — Метод изменения имени у объекта;
  - метод get_name — Метод получения имени объекта;

- метод get_header — Метод получения указателя на родительский объект;
- метод print_hierarchy — Метод вывода иерархии объекта на экран;
- метод get_child — Метод получения указателя на подчинённый объект по имени объекта.

Класс cl_obj:

- функционал:
  - метод cl_obj — Параметризированный конструктор.

Класс cl_application:

- функционал:
  - метод cl_application — Параметризированный конструктор;
  - метод build_tree_objects — Метод построения дерева объектов;
  - метод exec_app — Метод запуска программы.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс объекта в дереве	
		cl_application	public		3
2	cl_obj			Класс наследник класса cl_base для использования в структуре дерева	
		cl_obj	public		2
3	cl_application			Класс наследник класса cl_base используемый как изначальный объект в структуре дерева и для запуска программы	

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 *header - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Определение поля name параметром name	2
2		Определение поля header_ptr параметром header	3
3	header_ptr != nullptr	Добавление данного объекта в список children_ptr у объекта header_ptr	∅
			∅

3.2 Алгоритм метода edit_name класса cl_base

Функционал: Метод изменения имени у объекта.

Параметры: string new_name - имя для установки.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода edit_name класса cl_base

№	Предикат	Действия	№ перехода
1		Установка поля name параметром new_name	2
2		Возврат true	∅

3.3 Алгоритм метода get_name класса cl_base

Функционал: Метод получения имени объекта.

Параметры: нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля name	∅

3.4 Алгоритм метода get_header класса cl_base

Функционал: Метод получения указателя на родительский объект.

Параметры: нет.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_header класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля header_ptr	∅

3.5 Алгоритм метода print_hierarchy класса cl_base

Функционал: Метод вывода иерархии объекта на экран.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода print_hierarchy класса cl_base

№	Предикат	Действия	№ перехода
1	Вектор подчинённых объектов children_ptr не пустой	Вывод name	2
			∅
2		Инициализация итератора child типа cl_base*	3
3	child в children_ptr	Вывод двух пробелов и значения вызова метода get_name объекта child	4
			5
4		Шаг итерации child	3
5		Вызов метода print_hierarchy у последнего элемента вектора children_ptr	∅

3.6 Алгоритм метода get_child класса cl_base

Функционал: Метод получения указателя на подчинённый объект по имени объекта.

Параметры: string name - имя объекта для поиска.

Возвращаемое значение: cl_base* - указатель на подчинённый объект.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_child класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация итератора child типа cl_base*	2
2	child в children_ptr		3

№	Предикат	Действия	№ перехода
			4
3	Значение поля name у объекта child == name	Возврат child	∅
		Шаг итерации child	2
4		Возврат nullptr	∅

3.7 Алгоритм конструктора класса cl_obj

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl_obj

№	Предикат	Действия	№ перехода
1		Передача header и name как параметры в конструктор родительского класса cl_base	∅

3.8 Алгоритм конструктора класса cl_application

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - объект родительского класса.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		Передача header как параметр в конструктор родительского класса cl_base	∅

3.9 Алгоритм метода build_tree_objects класса cl_application

Функционал: Метод построения дерева объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление переменных parentName, childName типа string	2
2		Инициализация указателя parent на класс cl_base данным объектом	3
3		Инициализация указателя child на класс cl_base нулевым указателем	4
4		Ввод знаения parentName	5
5		Вызов метода edt_name с передачей переменной parentName как параметра	6
6		Ввод значения parentName и childName	7
7	parentName == childName		∅
			8
8	child != nullptr && parent != значение вызова метода get_name у объекта child	parent = child	9
			9
9	значение вызова метода get_child с парамером childName == nullptr && parentName == значению вызова метода get_name у	Присваивание child значения выделения памяти для класса cl_obj с передачей parent и childName как параметров в конструктор	6

№	Предикат	Действия	№ перехода
	объекта parent		
			6

3.10 Алгоритм метода exec_app класса cl_application

Функционал: Метод запуска программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод значения вызова метода get_name	2
2		Вызов метода print_hierarchy	3
3		Возврат значения 0	∅

3.11 Алгоритм функции main

Функционал: Главная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 12.

Таблица 12 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Инициализация объекта ob_cl_application класса cl_application с передачей nullptr в параметризированный конструктор	2
2		Вызов метода build_tree_objects у объекта ob_cl_application	3
3		Возврат значения вызова метод exec_app у объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

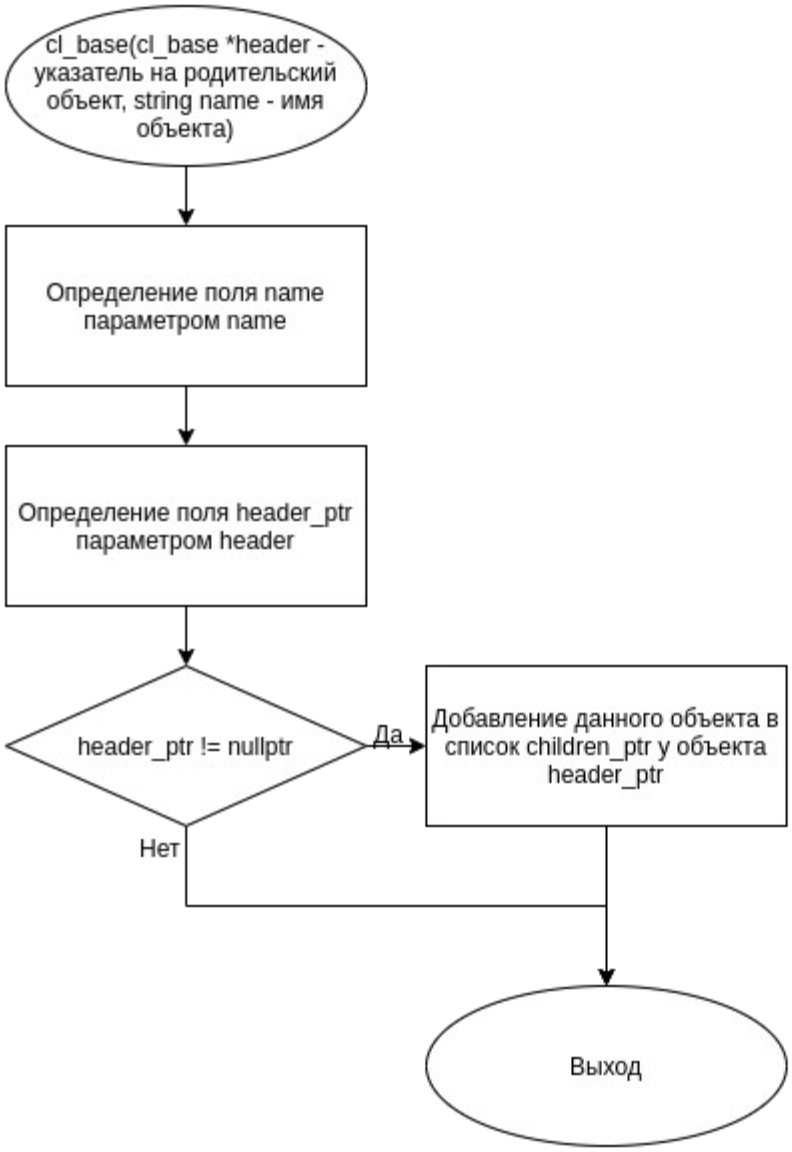

Рисунок 1 – Блок-схема алгоритма

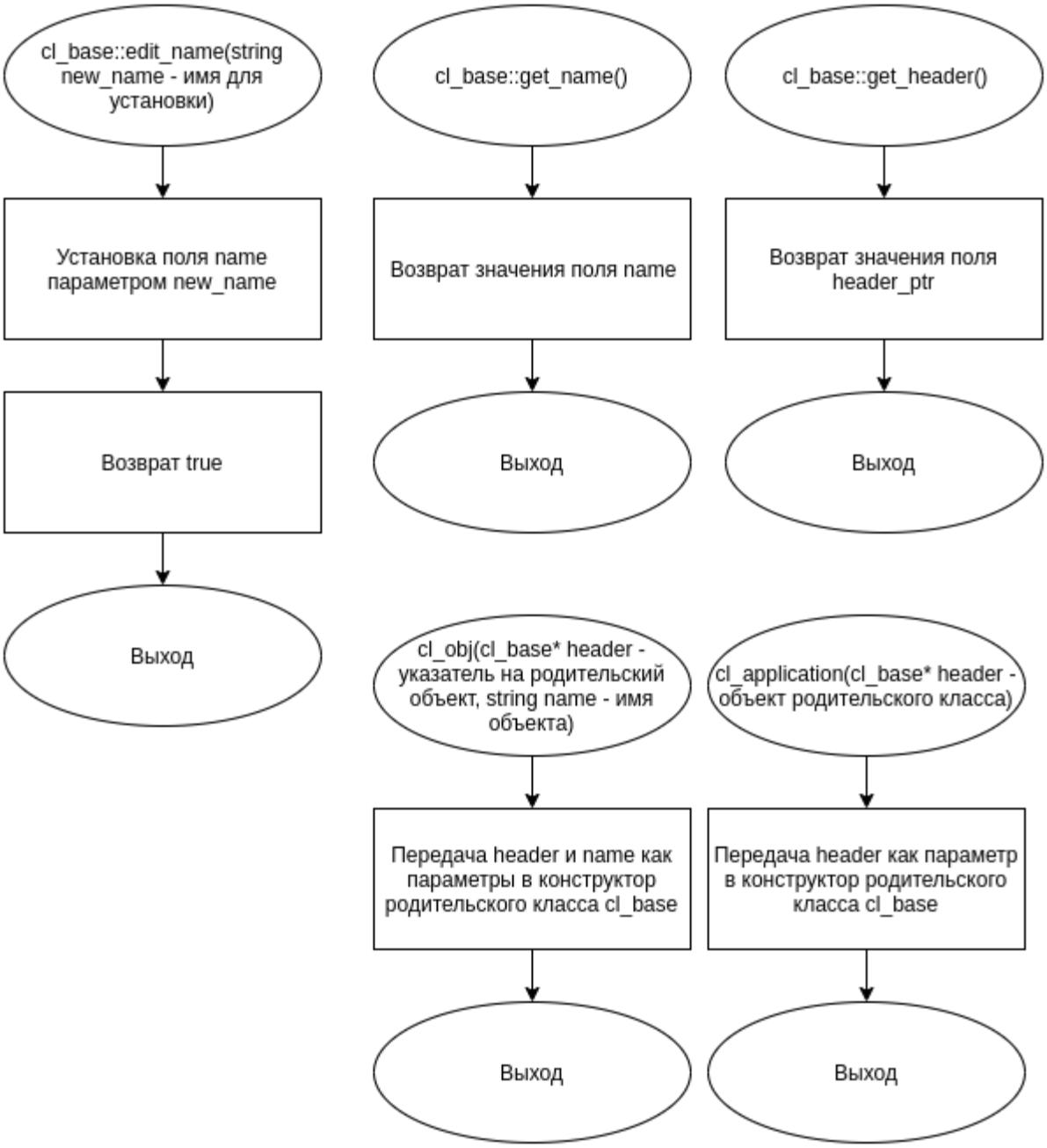

Рисунок 2 – Блок-схема алгоритма

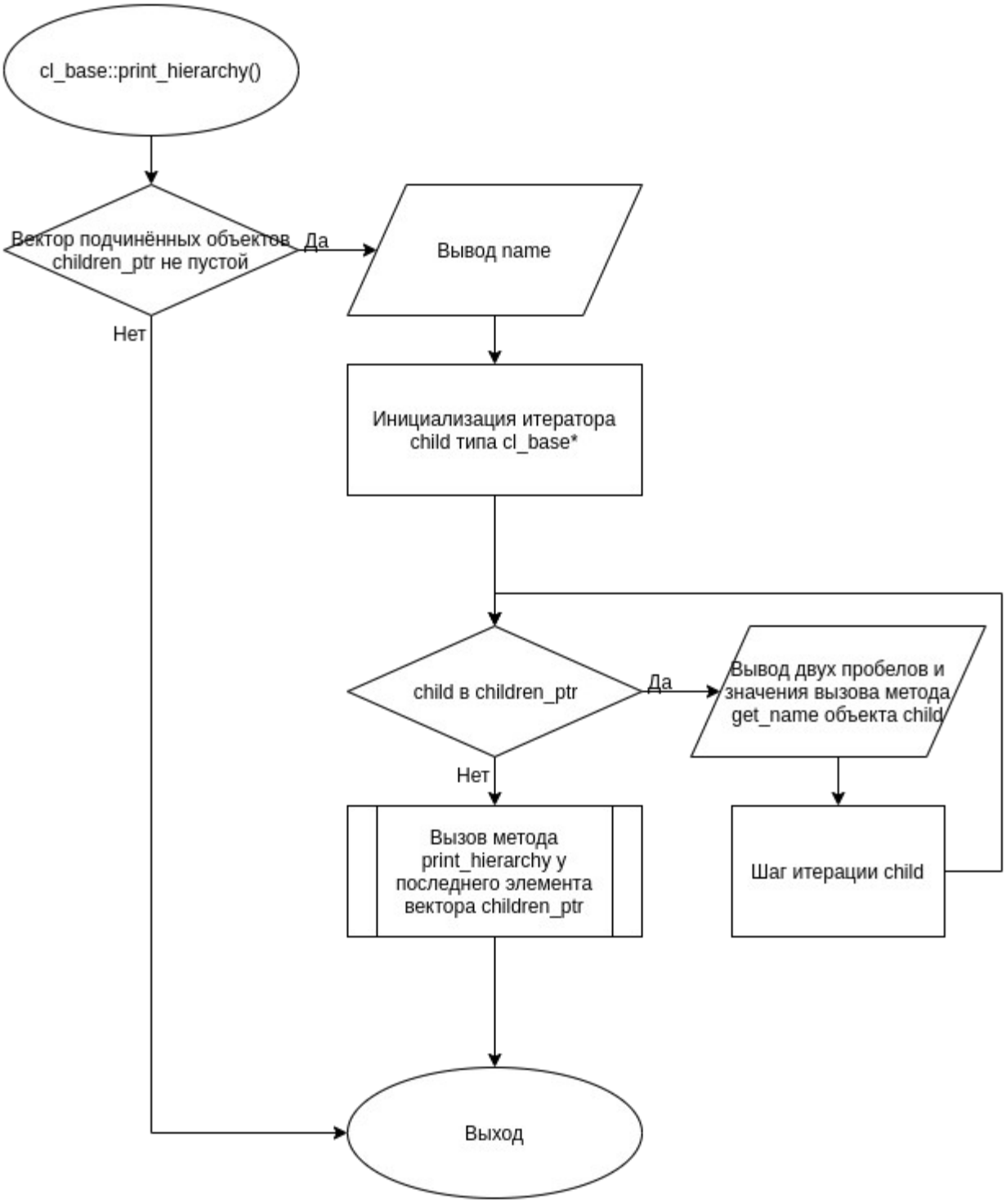

Рисунок 3 – Блок-схема алгоритма

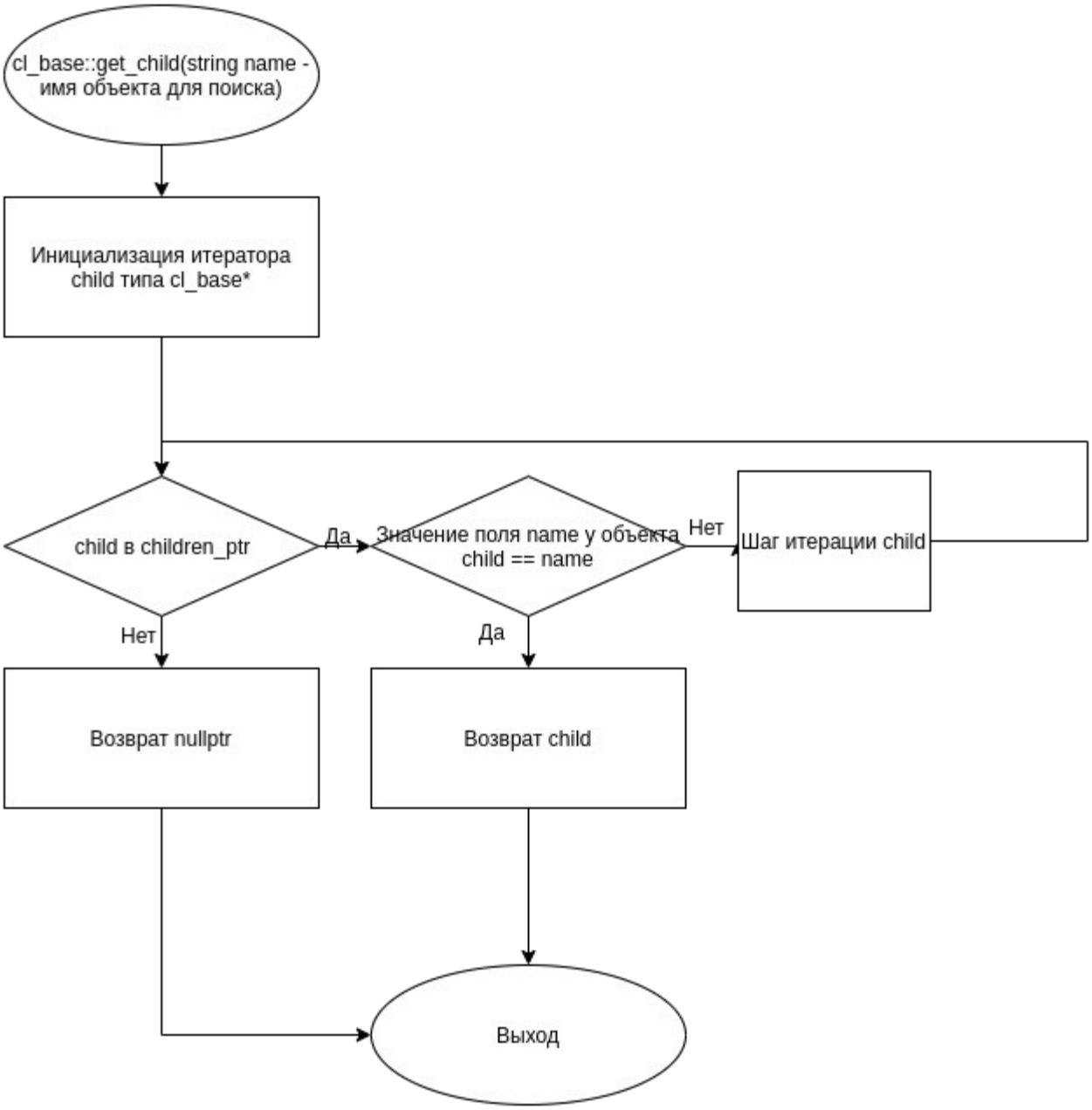

Рисунок 4 – Блок-схема алгоритма

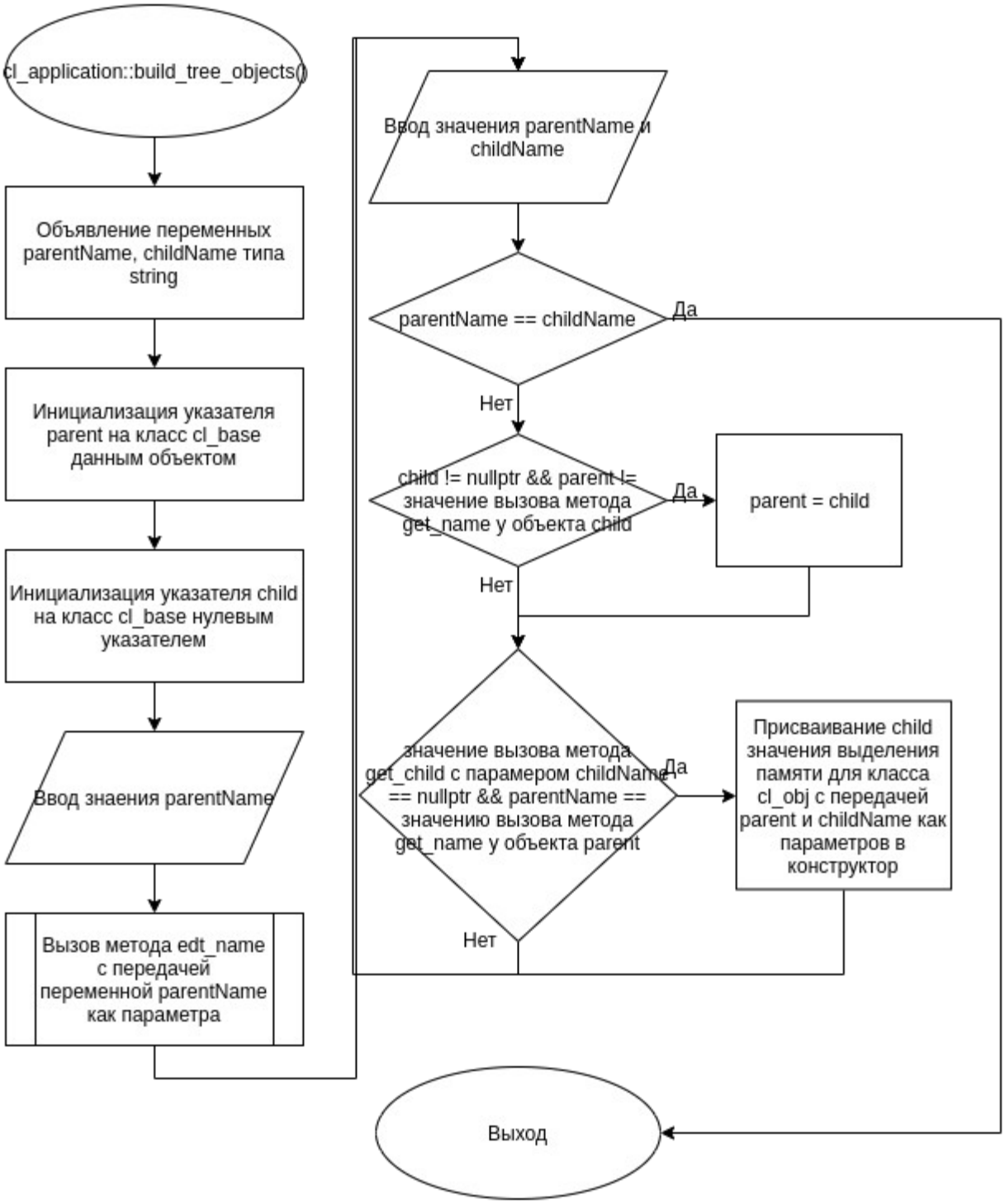

Рисунок 5 – Блок-схема алгоритма

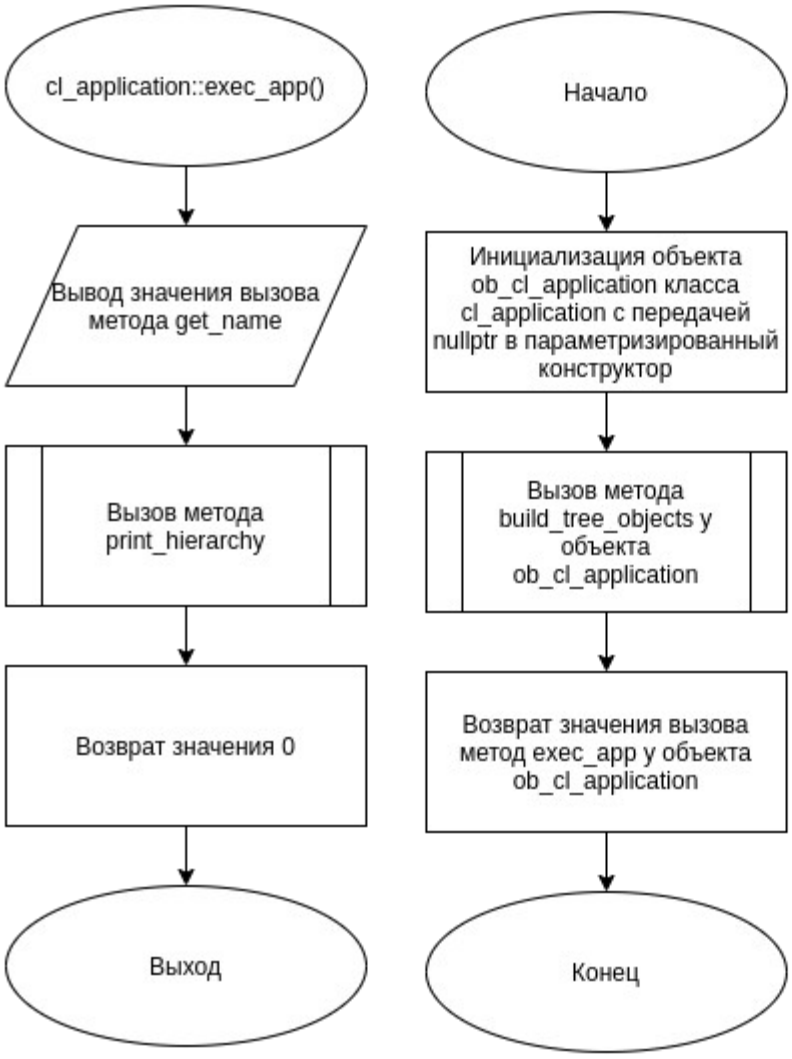

Рисунок 6 – Блок-схема алгоритма

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* header) : cl_base(header) {};
void cl_application::build_tree_objects() {
   string parentName, childName;
   cl_base *parent = this;
   cl_base *child = nullptr;

   cin >> parentName;
   this->edit_name(parentName);
   while(true)
   {
      cin >> parentName >> childName;
      if(parentName == childName) break;
      if(child != nullptr && parentName == child->get_name())
      {
         parent = child;
      }
      if(parent->get_child(childName) == nullptr && parentName == parent-
>get_name())
      {
         child = new cl_obj(parent, childName);
      }
   }

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

5.2 Файл cl_application.h

Листинг 2 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
#include "cl_obj.h"

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

#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"


cl_base::cl_base(cl_base* header, string name){
   this->name = name;
   this->header_ptr = header;
   if(header_ptr){
      header_ptr-> children_ptr.push_back(this);
   }
}

bool cl_base::edit_name(string new_name){
   this->name = new_name;
   return true;
}

string cl_base::get_name(){
   return this->name;
}

cl_base* cl_base::get_header(){
   return this->header_ptr;
}

void cl_base::print_hierarchy(){
   if(!children_ptr.empty())
```

```
        {
           cout << endl << name;
           for(cl_base* child : children_ptr) {
              cout << "  " << child->get_name();
           }
           children_ptr[children_ptr.size() - 1]->print_hierarchy();
        }
     }

     cl_base* cl_base::get_child(string name){
        for(cl_base* child : children_ptr) {
           if(child->name == name) return child;
        }
        return nullptr;
     }
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

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


     using namespace std;

     class cl_base {
     protected:
        string name;
        cl_base *header_ptr;
        vector<cl_base*> children_ptr;
     public:
        cl_base(cl_base* base, string name = "Object");
        bool edit_name(string new_name);
        string get_name();
        cl_base* get_header();
        void print_hierarchy();
        cl_base* get_child(string name);
     };

     #endif
```

5.5 Файл cl_obj.cpp

Листинг 5 – cl_obj.cpp

```
#include "cl_obj.h"


cl_obj::cl_obj(cl_base* header, string name) : cl_base(header, name) {};
```

5.6 Файл cl_obj.h

Листинг 6 – cl_obj.h

```
#ifndef __CL_OBJ__H
#define __CL_OBJ__H
#include "cl_base.h"

class cl_obj : public cl_base {
public:
   cl_obj(cl_base* header, string name);
};

#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  корневого
объекта
   ob_cl_application.build_tree_objects  (  );            //  конструирование
системы, построение дерева объектов
   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

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).