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.

Создать базовый класс со следующими элементами:

Свойства:

- наименование объекта (строкового типа);
- указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно 0);
- массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.

Функционал:

- параметризированный конструктор с параметрами: указатель на головной объект в дереве иерархии и наименование объекта (имеет значение по умолчанию);
- метод определения имени объекта;
- метод получения имени объекта;
- метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- метод переопределения головного объекта для текущего в дереве иерархии (не допускается задание головного объекта для корневого и появление второго корневого объекта);

- метод получения указателя на головной объект текущего 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 программы-системы, изделия);

− 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. Каждый объект имеет уникальное имя.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

```
int main()
{
   cl_application  ob_cl_application ( nullptr );
   ob_cl_application.bild_tree_objects ( );  // 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.

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.

Примерввода

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

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	Base	cl_1	public	Базовый класс содержит основные поля и методы	2	
		Application	public		3	
2	cl_1			Производный класс, наследуемый от класса base		
3	Application			Класс корневого объекта (приложения)		

Для решения задачи используются:

Класс base (базовый класс): Объект стандартного потока вывода на экран

cout.

Объект стандартного ввода с клавиатуры cin.

Условный оператор if.

Объект класса ob_application класса application.

Объекты класса cl_1 (имена и количество задаются пользователем).

Свойства/поля:

Поле, отвечающее за наименование объекта

Наименование - object_name;

Тип - строковый;

Модификатор доступа - private.

Поле, отвечающее за указатель на головной объект для текущего объекта .

Наименование - p_parent;

Тип - указатель на объект класса base (или его наследников);

Модификатор доступа - private.

Поле, отвечающее за состояние объекта

Наименование - state;

Тип - целочисленный;

Модификатор доступа - private;

Поле, отвечающее за массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии:

Наименование - children;

Тип - вектор ;

Модификатор доступа - private.

Методы:

Конструктор base:

Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта (имеет значение по умолчанию).

set_object_name:

Функционал - используется для установки имени объекта.

get_object_name:

Функционал - используется для получения имени объекта.

change_parent:

Функционал - используется для переопределения головного объекта для текущего в дереве иерархии.

get_parent:

Функционал - используется для получения указателя на головной объект текущего объекта.

set_state:

Функционал - используется для определения состояние объекта.

get_state:

Функционал - используется для получения состояние объекта.

get_child_by_name:

Функционал - используется для получения указателя объект-потомок по имени объекта

print_object_tree:

Функционал - используется для вывода наименований объектов в дереве иерархии слева направо и сверху вниз.

Деструктор ~base:

Функционал - деструктор, удаление объектов иерархического дерева.

Класс application (наследуется от base)

Методы:

Конструктор application:

Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии

bild_tree_objects:

Функционал - используется для построения исходного дерева иерархии объектов

exec_app:

Функционал - метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Класс cl_1 (наследуется от base):

Методы

Конструктор cl_1

Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.0 Алгоритм метода Exec_App класса App

Функционал: Запуск приложения.

Параметры: Нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода Exec_App класса App

№	Предикат	Действия	№ перехода
1		Вызов метода Print() текущего объекта	2
2		Возврат 0	∅

3.1 Алгоритм метода Build_Tree_Objects класса App

Функционал: Построение дерева иерархии.

Параметры: Нет.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода Build_Tree_Objects класса App

№	Предикат	Действия	№ перехода
1		Инициализация указателя temp типа Derived* значением nullptr	2

№	Предикат	Действия	№ перехода
		Объявление string n_a, n_b	
2		Ввод (n_a)	3
3		Вызов метода SetName(n_a) для текущего объекта	4
4		Ввод(n_a, n_b)	5
5	n_a != n_b		6
			8
6	n_a == имя текущего объекта	Запись в temp объекта типа Derived путем вызова конструктора с параметрами (this, n_b)	8
			7
7	n_a == имя temp	Запись в temp объекта Derived путем вызова конструктора с параметрами (temp, n_b)	8
		Запись в temp объекта Derived путем вызова конструктора с параметрами (головной элемент для temp, n_b)	8
8	n_a != n_b		4
			∅

3.2 Алгоритм конструктора класса base

Функционал: параметризированный конструктор, устанавливается имя объекта и головной объект для текущего.

Параметры: p_parent - указатель на объект класса base (указатель на головной объект в дереве иерархии); object_name - строковый, наименование объекта.

Алгоритм конструктора представлен в таблице 4.

Таблица 4 – Алгоритм конструктора класса base

№	Предикат	Действия	№ перехода
1		Присвоение полю	2

№	Предикат	Действия	№ перехода
		object_name текущего объекта значение параметра object_name	
2		Присвоение полю p_parent текущего объекта значение параметра p_parent	3
3		Присвоение полю state текущего объекта значения 0	4
4	Указатель на головной объект p_parent для текущего объекта не нулевой	Добавление для головного объекта текущий объект в массив подчиненных объектов	∅
			∅

3.3 Алгоритм метода get_object_name класса base

Функционал: используется для получения имени объекта.

Параметры: Нет.

Возвращаемое значение: строковый, имя объекта.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_object_name класса base

№	Предикат	Действия	№ перехода
1		возврат имени текущего объекта object_name	∅

3.4 Алгоритм метода change_parent класса base

Функционал: используется для переопределения головного объекта для текущего в дереве иерархии.

Параметры: указатель на объект класса base, new_parent - указатель на новый головной объект для текущего объекта.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода change_parent класса base

№	Предикат	Действия	№ перехода
1	Указатель на головной объект для текущего объекта равен нулевому указателю или значение параметра new_parent равно нулевому указателю		∅
		Инициализация целочисленной переменной счетчика i значением 0	2
2		Добавление в текущего элемента в массив наследников объекта newhead	3
3	Значение переменной счетчика меньше размера массива	Запись в поле head текущего объекта значения адреса newhead	4

№	Предикат	Действия	№ перехода
	подчиненных объектов children для текущего головного		
			5
4	Значение i-ого элемента массива children текущего головного равно указателю на текущий объект	удаление текущего объекта из списка наследников объектародителя	5
		увеличение переменной счетчика i на 1	3
5		Присвоение полю p_parent текущего объекта значения параметра new_parent	6
6		добавление текущего объекта в массив наследников нового головного объекта new_parent	∅

3.5 Алгоритм метода Print класса BaseClass

Функционал: Вывод дерева иерархии.

Параметры: Нет.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода Print класса BaseClass

№	Предикат	Действия	№ перехода
1		Инициализация указателя curr типа BaseClass* значением адреса текущего объекта	2
2	Головной элемент объектта curr!= nullptr	Запись указателя на головной элемент в curr	2
			3
3		Вывод имени curr	4
4	Массив наследников curr не пустой	Переход на новую строку	5
			5
5	Массив наследников curr не пустой	Вывод имени curr	6
			∅
6		Инициализация int i = 0	7
7	i меньше размера массива наследников curr	Форматированный вывод i-го элемента массива наследников curr	8
			9
8		Инкремен i	7
9		Запись в curr последнего элемента из массива наследников	10
10	Массив наследников curr не пустой	Переход на новую строку	5
			5

3.6 Алгоритм метода GetNAme класса BaseClass

Функционал: Получение имени объекта.

Параметры: Нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода GetNAme класса BaseClass

№	Предикат	Действия	№ перехода
1		Возврат name	∅

3.7 Алгоритм метода SetName класса BaseClass

Функционал: Определение имени объекта.

Параметры: name.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода SetName класса BaseClass

№	Предикат	Действия	№ перехода
1		Запись в поле name объекта значения полученного параметра name	∅

3.8 Алгоритм функции main

Функционал: Точка входа в программу.

Параметры: Нет.

Возвращаемое значение: int - код завершения программы.

Алгоритм функции представлен в таблице 10.

Таблица 10 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта класса ob_application посредством параметризированного	2

№	Предикат	Действия	№ перехода
		конструктора с параметром нулевого указателя	
2		вызов метода bild_tree_objects текущего объекта	3
3		возращение результата вызова метода exec_app текущего объекта	∅

3.9 Алгоритм конструктора класса Derived

Функционал: Создание объекта класса Derived.

Параметры: BaseClass* b, string n.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса Derived

№	Предикат	Действия	№ перехода
1		Вызов параметрезированного конструктора родительского класса BaseClass с параметрами b, n	∅

3.10 Алгоритм конструктора класса App

Функционал: Создание объекта класса App.

Параметры: Derived* p.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса App

№	Предикат	Действия	№ перехода
1		Передача параметров p и "root" в конструктор родительского класса BaseClass	∅

3.11 Алгоритм деструктора класса BaseClass

Функционал: Деструктор.

Параметры: Нет.

Алгоритм деструктора представлен в таблице 13.

Таблица 13 – Алгоритм деструктора класса BaseClass

№	Предикат	Действия	№ перехода
1		Инициализация int i = 0	2
2	i < размер массива указателей на наследников	Освобождение памяти, выделенной под i-ый элемент памяти	3
			∅
3		Инкремент i	2

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-4.

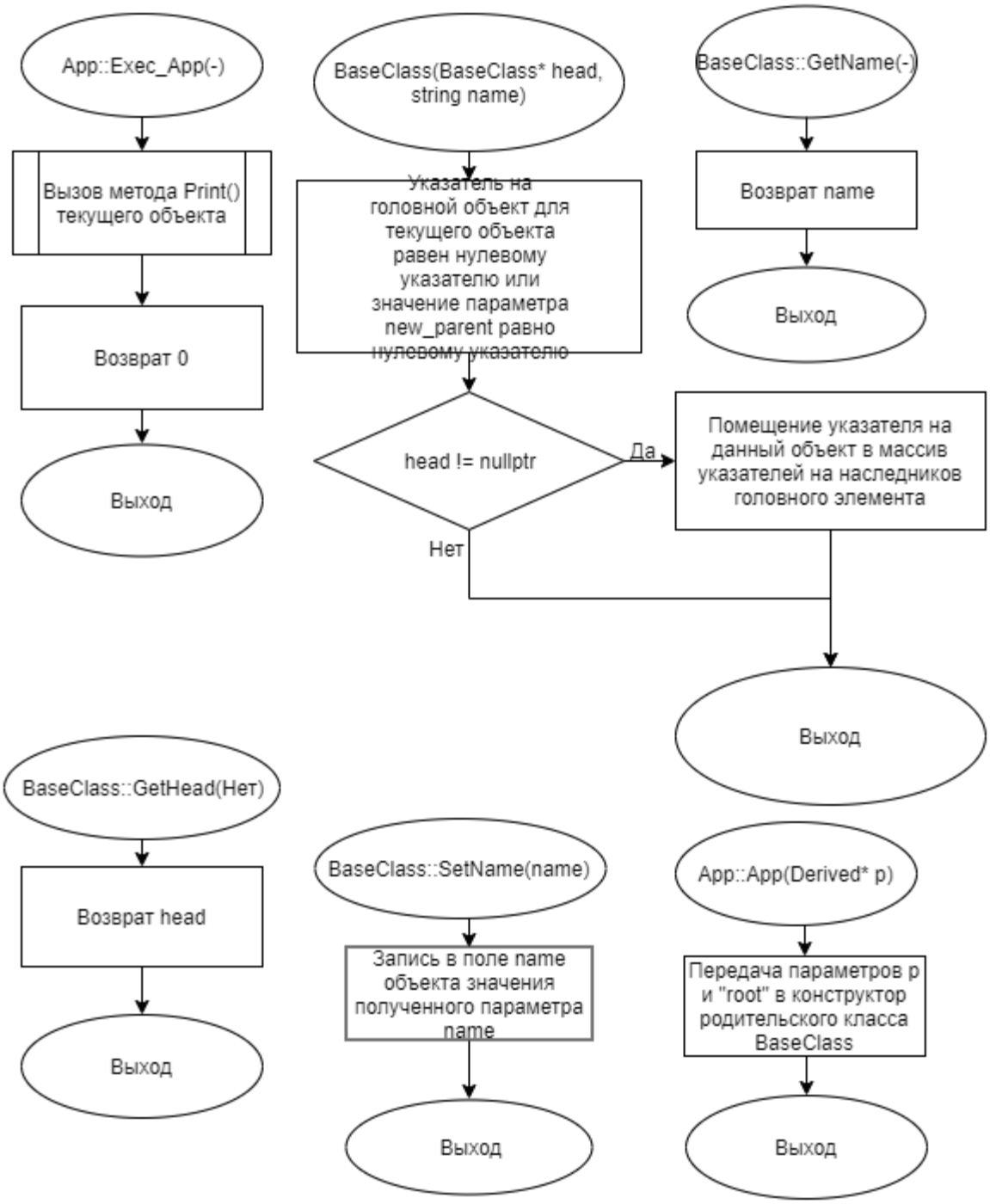

Рисунок 1 – Блок-схема алгоритма

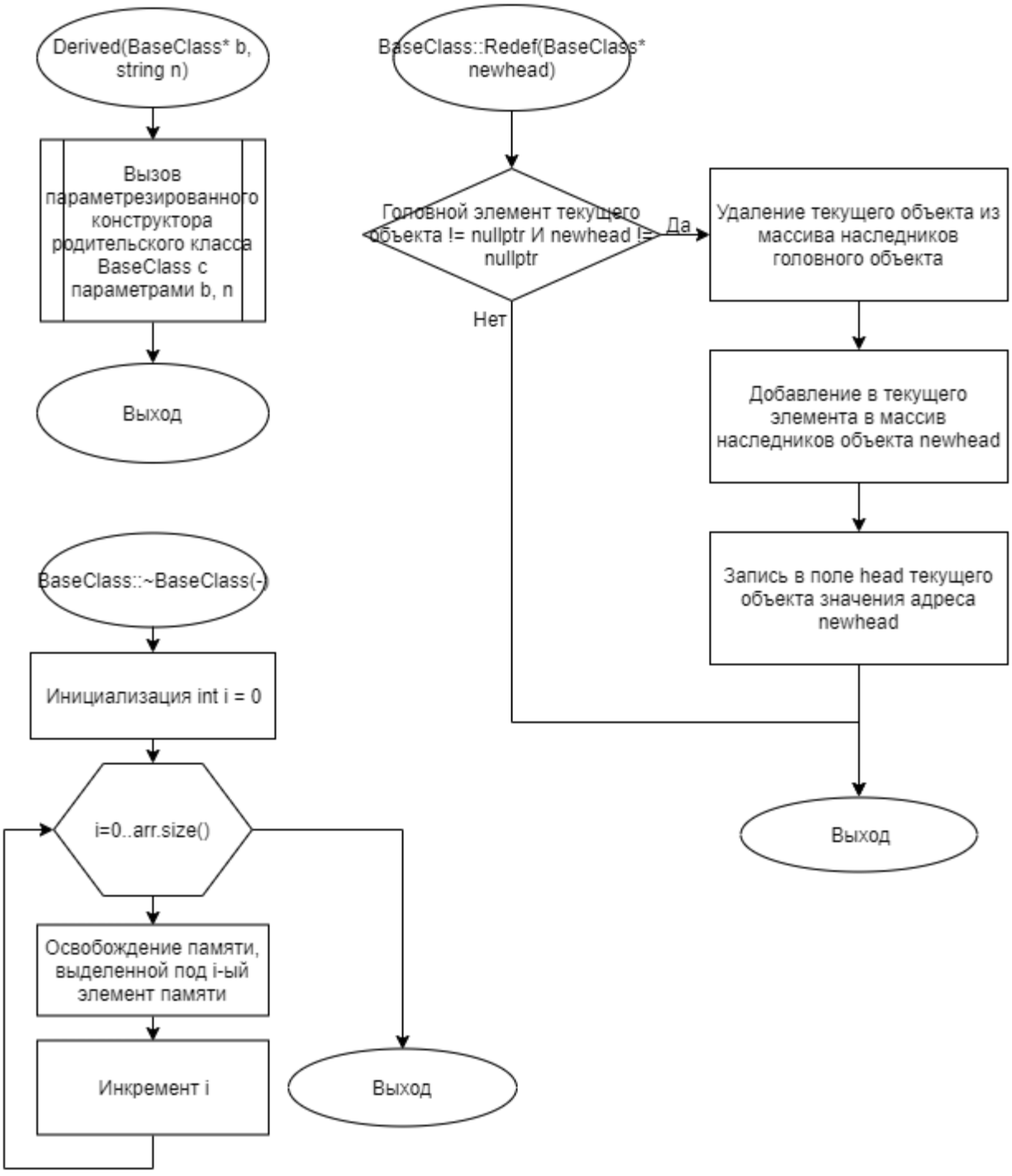

Рисунок 2 – Блок-схема алгоритма

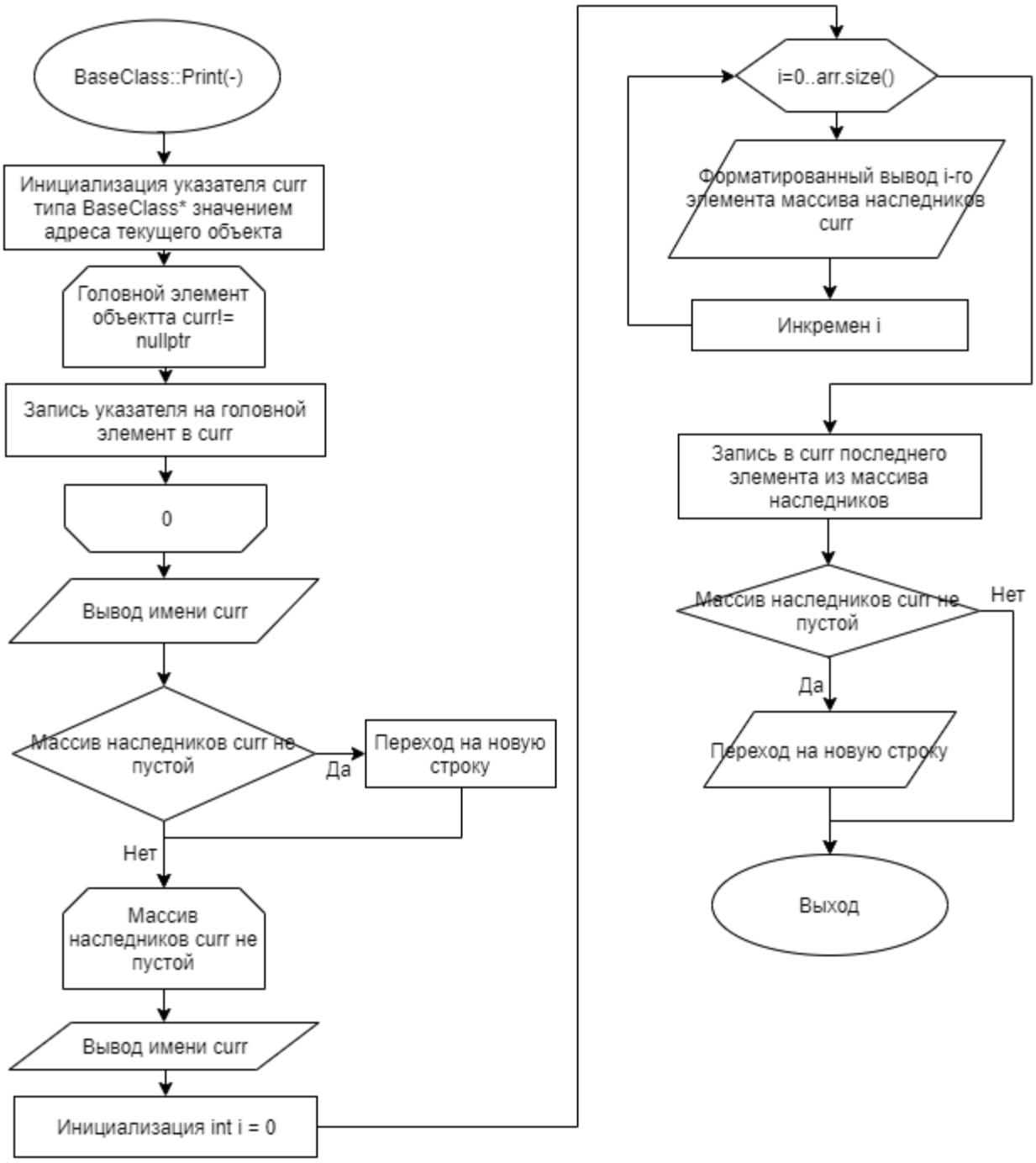

Рисунок 3 – Блок-схема алгоритма

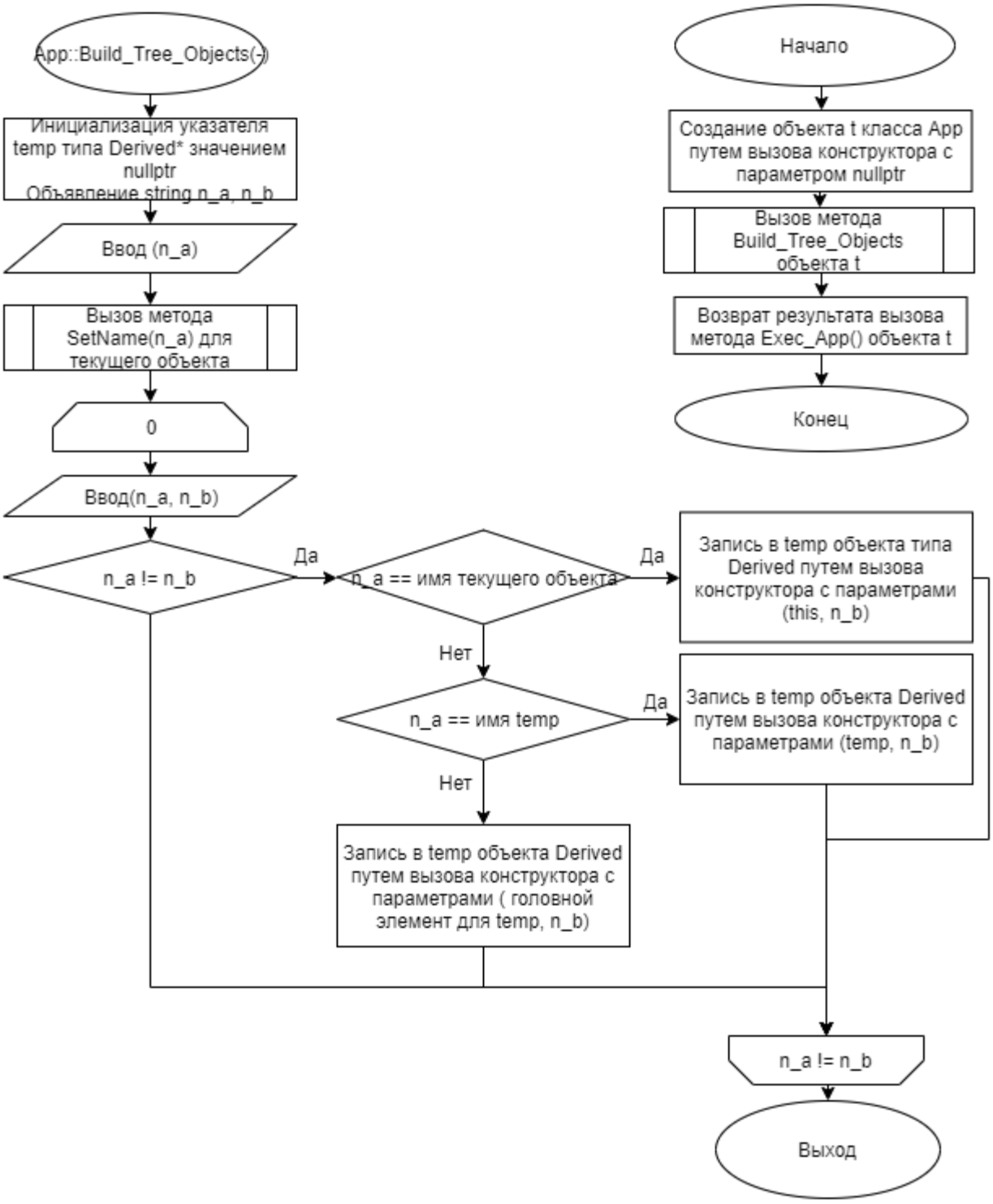

Рисунок 4 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.0 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
application::application(base*   p_parent)   :base(p_parent)//конструктор   класса
приложение
{}

void application::bild_tree_objects()
{
	string name_one, name_two;
	cin >> name_one;
	this->set_object_name(name_one);
	base* obj_one = this;

	while (cin >> name_one >> name_two)
	{
		if (name_one == name_two) return;
		else
		{
			if (obj_one->get_object_name() == name_one)
			{
				base* obj_two = new cl_1(obj_one,
					name_two);
				obj_one = obj_two;

			}
			else
			{
				base* parent = obj_one->get_parent();
				base* obj_two = new cl_1(parent,
					name_two);
				obj_one = obj_two;

			}
		}
	}
}


int application::exec_app()
{
	return print_object_tree();
```

```
}
```

5.1 Файл application.h

Листинг 2 – application.h

```
#ifndef APPLICATION_H_
#define APPLICATION_H_
#include <string>
#include "base.h"
#include "cl_1.h"

class application : public base
{
public:
application(base* p_parent);
void bild_tree_objects();// метод построения дерева иерархии
int exec_app();//метод запуска приложения
};
#endif
```

5.2 Файл base.cpp

Листинг 3 – base.cpp

```
#include "base.h"
base::base(base* p_parent, string object_name)//конструктор базового класса
{
	this->object_name = object_name;
	this->p_parent = p_parent;
	state = 0;//отметка неготовности объекта
	if (p_parent)
	{
		p_parent->children.push_back(this);
	}
}
void base::set_object_name(string object_name)
{
	this->object_name = object_name;
}
string base::get_object_name()
{
	return object_name;
}
void base::change_parent(base * new_parent)//переопределить головной объект для текущего
{
	if (this->p_parent == nullptr || new_parent == nullptr) return;
```

```
	for (int i = 0; i < p_parent->children.size(); i++)
	{
		if (p_parent->children[i] == this)
		{
			p_parent->children.erase(p_parent->children.begin()
				+ i);//удаление текущего объекта из списка наследников объекта-родителя
			return;
		}
	}
	this->p_parent = new_parent;//изменение у текущего объекта указатель на головной объект
		new_parent->children.push_back(this);//добавление текущего объекта в список наследников нового головного объекта
}
base* base::get_parent() //получение указателя на головной объект
{
	return p_parent;
}

void base::set_state(int state)//определить состояние объекта
{
	this->state = state;
}
int base::get_state()//получить состояние объекта
{
	return state;
}
base* base::get_child_by_name(string object_name)
{
	if (children.size() == 0) return 0;
	for (int i = 0; i < children.size(); i++)
	{
		if (children[i]->get_object_name() == object_name)
		{
			return children[i];
		}
	}
	return 0;
}
int base::print_object_tree()
{
	base* ob_parent = this;
	if (this->p_parent == nullptr) cout << this->get_object_name();
	while (ob_parent->children.size())
	{
		int i = 0;
		cout << endl << ob_parent->get_object_name();
		for (i; i < ob_parent->children.size(); i++)
		{
			cout << "  " << ob_parent->children[i] -> get_object_name();
		}

		ob_parent = ob_parent->children[i - 1];
	}
	return 0;
}
```

```
base::~base()
{
	for (int i = 0; i < children.size(); i++)
	{
		delete children[i];
	}

}
```

5.3 Файл base.h

Листинг 4 – base.h

```
#ifndef BASE_H_
#define BASE_H_
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class base
{
private:
string object_name;//наименование объекта
base* p_parent;//указатель на головной объект
int state; //состояние объекта
vector <base*>children;//массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии
public:
base( base* p_parent , string object_name = "base" );
void set_object_name(string object_name);
string get_object_name();//получить имя объекта
void change_parent( base* new_parent);//переопределить головной объект для текущего
base* get_parent();//получить указатель на головной объект для текущего
void set_state (int state);//определить состояние объекта
int get_state ();//получить состояние объекта
base* get_child_by_name(string object_name);//получить указатель на объект потомок по имени объекта
int print_object_tree();
~base();//деструктор
};
#endif
```

5.4 Файл cl_1.cpp

Листинг 5 – cl_1.cpp

```
#include "cl_1.h"
cl_1::cl_1(base* p_parent, string object_name):base(p_parent,
```

```
object_name)
{}
```

5.5 Файл cl_1.h

Листинг 6 – cl_1.h

```
#ifndef CL_1_H
#define CL_1_H
#include "base.h"
class cl_1: public base
{
public:
cl_1(base* p_parent, string object_name);
};
#endif
```

5.6 Файл main.cpp

Листинг 7 – main.cpp

```
#include "application.h"
int main()
{
application ob_application(nullptr);//создание объекта класса application
ob_application.bild_tree_objects();//построение дерева иерархии
return ob_application.exec_app();//запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 14.

Таблица 14 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
Object_rootObject_rootObject_1Object_rootObject_2Object_rootObject_3Object_3 Object_4Object_5Object_6Object_6	Object_rootObject_rootObject_1Object_rootObject_2Object_rootObject_3Object_3	Object_rootObject_rootObject_1Object_rootObject_2Object_rootObject_3Object_3
obj_root obj_root obj_1 obj_1 obj_2 obj_1 obj_3 obj_1 obj_4 obj_4 obj_5 obj_6 obj_6	obj_root obj_root obj_1 obj_1 obj_2 obj_3 obj_4 obj_4 obj_5	obj_root obj_root obj_1 obj_1 obj_2 obj_3 obj_4 obj_4 obj_5
Object_root Object_root Object_1 Object_root ObJect_2 Object_3 Object_4 Object_3 Object_5 Object_6 Object_6	Object_root Object_root Object_1 ObJect_2 Object_4 Object_5	Object_root Object_root Object_1 ObJect_2 Object_4 Object_5
obj_root obj_root obj_1 obj_1 obj_2 obj_2 obj_3 obj_3 obj_4 obj_3 obj_5 obj_5 obj_5	obj_root obj_root obj_1 obj_1 obj_2 obj_2 obj_3 obj_3 obj_4 obj_5	obj_root obj_root obj_1 obj_1 obj_2 obj_2 obj_3 obj_3 obj_4 obj_5

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. 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).

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).