МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

УЧЕБНОЕ ЗАДАНИЕ

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« Задание 4_1_1 »

С тудент группы	ИКБО-01-21	Резников Г.А.
Руководитель практики	Ассистент	Данилович Е.С.
Работа представлена	«___»_____ 2022 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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;
- 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.

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

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

Метод решения

Использование объектов потока ввода-вывода cin и cout, объектов стандартной библиотеки string и vector.

Использование объекта q класса app. При этом класс app является наследником базового класса base.

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	base			Базовый класс в иерархии наследования, содержит основные поля и методы		
		app	public		2	
		base_c1	public		3	
2	app			Класс приложения, необходим для запуска, функционирования программы, а также осуществления поставленной задачи		
3	base_c1			Класс объекта, с которым работает app при взаимодействии с деревом		

Класс base:

- Свойства/поля:

- Поле, отвечающее за хранение имени объекта
    - Имя - name
    - Тип - string
    - Модификатор доступа - protected
- Поле, отвечающее за хранение указателя на головной объект для данного объекта
    - Имя - head
    - Тип - base*
    - Модификатор доступа - protected
- Поле, хранящее массив указателей на подичненные элементы
    - Имя - ar_p
    - Тип - vector<base*>
    - Модификатор доступа - protected

- Функционал:
    - Метод base - конструктор, создает новый объект и принимает в качетсве параметра ссылку на головной объект для создаваемого и строковую переменную - имя создаваемого объекта (значение по умолчанию - "Def_name")
    - Метод set_name - устанавливает имя для текущего объекта, принимает в качестве параметра строковую переменную - устанавливаемое имя
    - Метод get_name - возвращает имя текущего объекта (строковое)
    - Метод display - вывод наименований объектов в дереве иерархии слева направо и сверху вниз
    - Метод rebase - переопределение головного объекта для текущего объекта с сохранением структуры дерева, принимает в качетсве

параметра указатель но новый головной объект

- Метод get_head_p - получения указателя на головной объект текущего объекта

Класс app:

- Функционал:
  - Метод app - конструктор - принимает в качетсве параметра ссылку (тип - ссылка base_c1) для корневого элемента на головной для него
  - Метод build_tree_oblects - метод построения дерева иерархии в соответствии с задачей
  - Метод exec_app - метод, отвечающий за выполнение поставленной задачи (вывод дерева иерархии в формате, заданном условием)

Класс base_c1:

- Функционал:
  - Метод base_c1 - конструктор, принимает в качестве параметров ссылку на головной элемент для создаваемного и его имя

Описание алгоритма

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

Функция: main

Функционал: Создвние объекта класса app, построение и вывод содержимого с помощбю соответствующих методов

Параметры: нет

Возвращаемое значение: int - код завершения программы

Алгоритм функции представлен в таблице 2.

Таблица 2. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		создание объекта q с передачей в контруктрор nullptr - указателя на головной объект для корневого объекта	2	
2		Построеение дерева иерархии вызовом метода build_tree_objects объекта q	3	
3		Возвртам метода exec_app объекта q	∅	

Класс объекта: base

Модификатор доступа: public

Метод: base

Функционал: Создание нового объекта в дереве иерархии

Параметры: base* h - ссылка на головной объект для создаваемого, string n - имя создаваемого объекта (значение по умолчанию - Def_name)

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода base класса base

№	Предикат	Действия	№ перехода	Комментарий
1		присвоить указателю на голову этого объекта ссылку на голову (параметр h)	2	
2		присвоить имени этого объекта передаваемое имя (параметр n)	3	
3	головной объект для создаваемого не nullptr	поместить созданный объект в конец массива подчиненных головному объекту объектов	∅	
			∅	

Класс объекта: base

Модификатор доступа: public

Метод: set_name

Функционал: Установка имени объекта

Параметры: string n - новое имя элемента

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода set_name класса base

№	Предикат	Действия	№ перехода	Комментарий
1		Присвоить полю name текущего объекта передаваемое значение n	∅	

Класс объекта: base

Модификатор доступа: public

Метод: get_name

Функционал: Возврат имени текущего объекта

Параметры: нет

Возвращаемое значение: string - имя текущего объекта

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода get_name класса base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат имени текущего объекта	∅	

Класс объекта: base

Модификатор доступа: public

Метод: display

Функционал: Вывод дерева иерархии

Параметры: нет

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода display класса base

№	Предикат	Действия	№ перехода	Комментарий
1		Инициализация указателя (тип base*) now = ссылка на текущий обхект, целочисленной переменной i = 0	2	
2	голова для now не nullptr	присвоить now указатель на голову для now	2	
			3	
3		Вывод имени now	4	
4	размер массива подчиненных now объектов не равен 0	переход на новую строку	5	
			5	
5	размер массива подчиненных now объектов не равен 0		6	
			∅	
6		Вывод имени now	7	
7	i меньше размера массива подчиненных now элементов	Вывод имени i-того элемента массива подчиненных now элементов	8	
			9	
8		увеличение i на 1	7	
9		Присвоить now последний элемент массива подчиненных now элементов	10	
10	размер массива	переход на новую строку	5	

	подчиненных now объектов не равен 0			
			5	

Класс объекта: base

Модификатор доступа: public

Метод: rebase

Функционал: переопределение головного элемента для текущего

Параметры: base* new_head - ссылка на новый головной объект для текущего объекта

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода rebase класса base

№	Предикат	Действия	№ перехода	Комментарий
1		инициализания логической is_in_tre = false, указателя типа base now = new_head	2	
2	головной элемент для now не nullptr	now присвоить указаетель на головной элемент для now	2	
			3	
3	размер ar_p у now не ноль		4	
			6	
4	текцщий элемент не содержится в	is_in_tree = false	5	

	массиве ar_p у now			
		is_in tree = true	6	
5		присвоить now адрес последнего элемента из массива ar_p при now	3	
6	головной элемент для текущего не nullptr И new_head не nullptr	удалить текущий элемент из массива говоного элемента для текущего	7	
			9	
7		полю head текущего элемента присвоить new_head	8	
8		добавить текущий элемент в массив подчиненных new_head элементов	∅	
9	головной элемент для текущего nullptr И is_in_tre = истина	удалить элемент new_head из массива говоного элемента для new_head	10	
			13	
10		полю head текущего элемента присвоить new_head	11	
11		полю head элемента new_head присвоить nullptr	12	
12		добавить текущий элемент в конец подчиненных элементов ar_p для new_head	∅	
13	головной элемент для текущего не nullptr И new_head равен nullptr	Инициализания переменной root типа base* ссылкой на текущий элемент	14	
			19	
14	головной элемент	root присвоить указатель	14	

	для root не равен nullptr	на головной элемент для root		
			15	
15		удалить текущий элемент из массива подчиненных головному для текущего элемента	16	
16		полю head элемента, на который ссылвется root присвоить адрес текущего элемента	17	
17		добавить в конец ar_p текущего элемента root	18	
18		полю head текущего элемента присвоить new_head	∅	
19	головной элемент для текущего равен nullptr И is_in_tre = ложь	полю head текущего элемента присвоить new_head	20	
			∅	
20		добавить текщий элемент в конец массива ar_p подчиненных new_head элементов	∅	

Класс объекта: base

Модификатор доступа: public

Метод: get_head_p

Функционал: Получение указателя на головной объект текущего объекта

Параметры: нет

Возвращаемое значение: base* - указатель на глоловной объект для текущего

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода get_head_p класса base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат head текущего объекта	∅	

Класс объекта: app

Модификатор доступа: public

Метод: app

Функционал: Конструктор - создает объект приложения app с передачей параметров p и строки "root" в конструктор родительского класса

Параметры: base_c1* p - указатель на головной объект для корневого объекта

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода app класса app

№	Предикат	Действия	№ перехода	Комментарий
1		Передать p в конструктор родительского класса base, а также строковое значение "root"	∅	

Класс объекта: app

Модификатор доступа: public

Метод: build_tree_objects

Функционал: Построение дерева иерархии

Параметры: нет

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода build_tree_objects класса app

№	Предикат	Действия	№ перехода	Комментарий
1		Инициализация переменной temp типа base_c1* = nullptr Объявление строковых переменных n_a и n_b	2	
2		Ввод n_a	3	
3		Вызов метода set_name для текущего элемента с передачей в качестве параметра n_a	4	
4		Ввод n_a и n_b	5	
5	n_a не равно n_b		6	
			8	
6	n_a равно имени текущего элемента	Создать объект типа base_c1 вызовом конструктора с передачей в качестве параметров адрес текущего элемента и n_b, и присвоить адрес созданного элемента переменной temp	8	
			7	
7	n_a равно имени temp	Создать объект типа base_c1 вызовом конструктора с передачей в качестве параметров temp и n_b, и присвоить адрес созданного элемента переменной temp	8	

		Создать объект типа base_c1 вызовом конструктора с передачей в качестве параметров ссылки на головной элемент для temp и n_b, и присвоить адрес созданного элемента переменной temp	8	
8	n_a не рано n_b		4	
			∅	

Класс объекта: app

Модификатор доступа: public

Метод: exec_app

Функционал: Вывод дерева иерархии

Параметры: нет

Возвращаемое значение: int - код завершения программы

Алгоритм метода представлен в таблице 11.

Таблица 11. Алгоритм метода exec_app класса app

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода display для текущего объекта	2	
2		Возврат 0	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

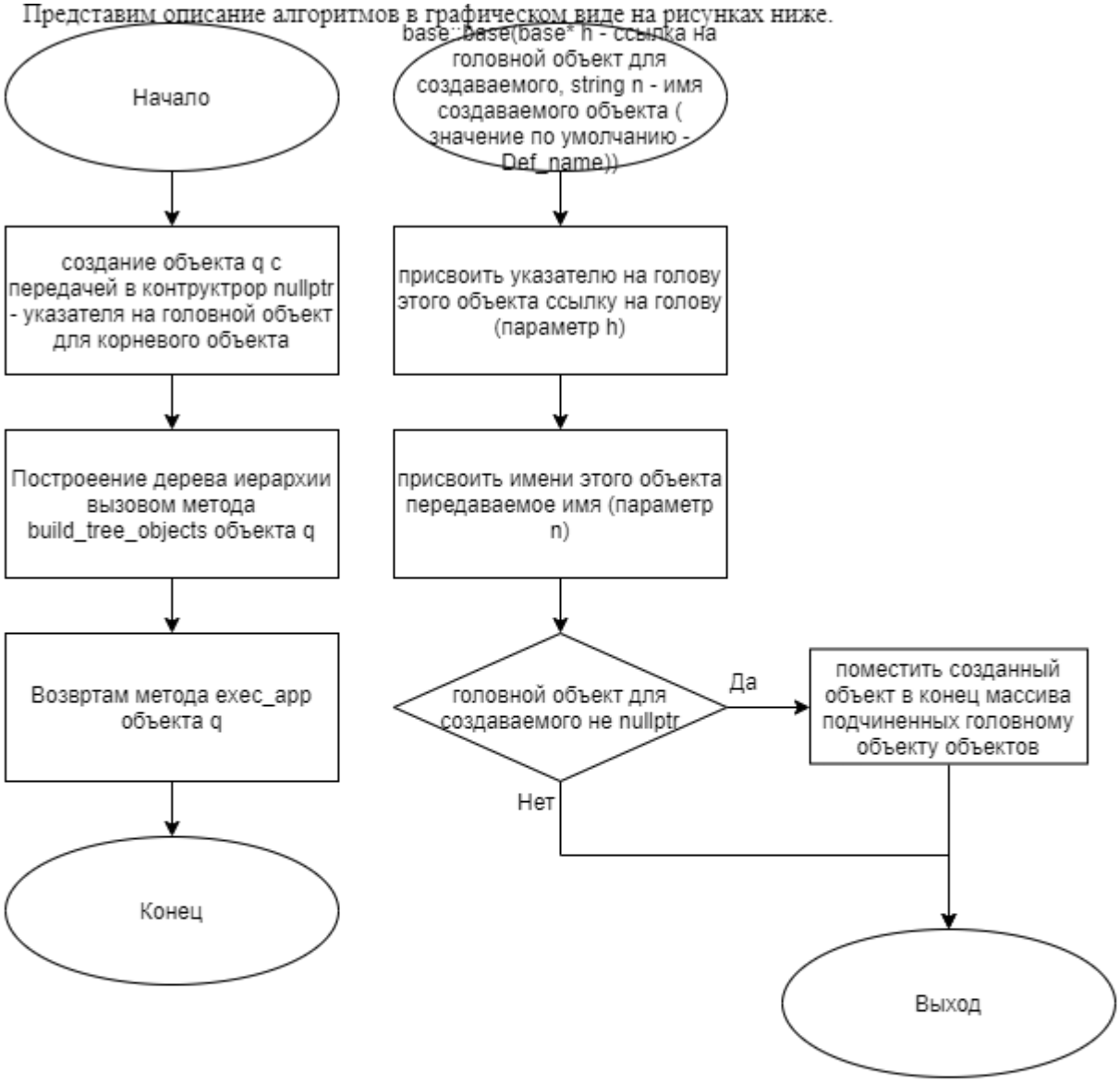

Рис. 1. Блок-схема алгоритма.

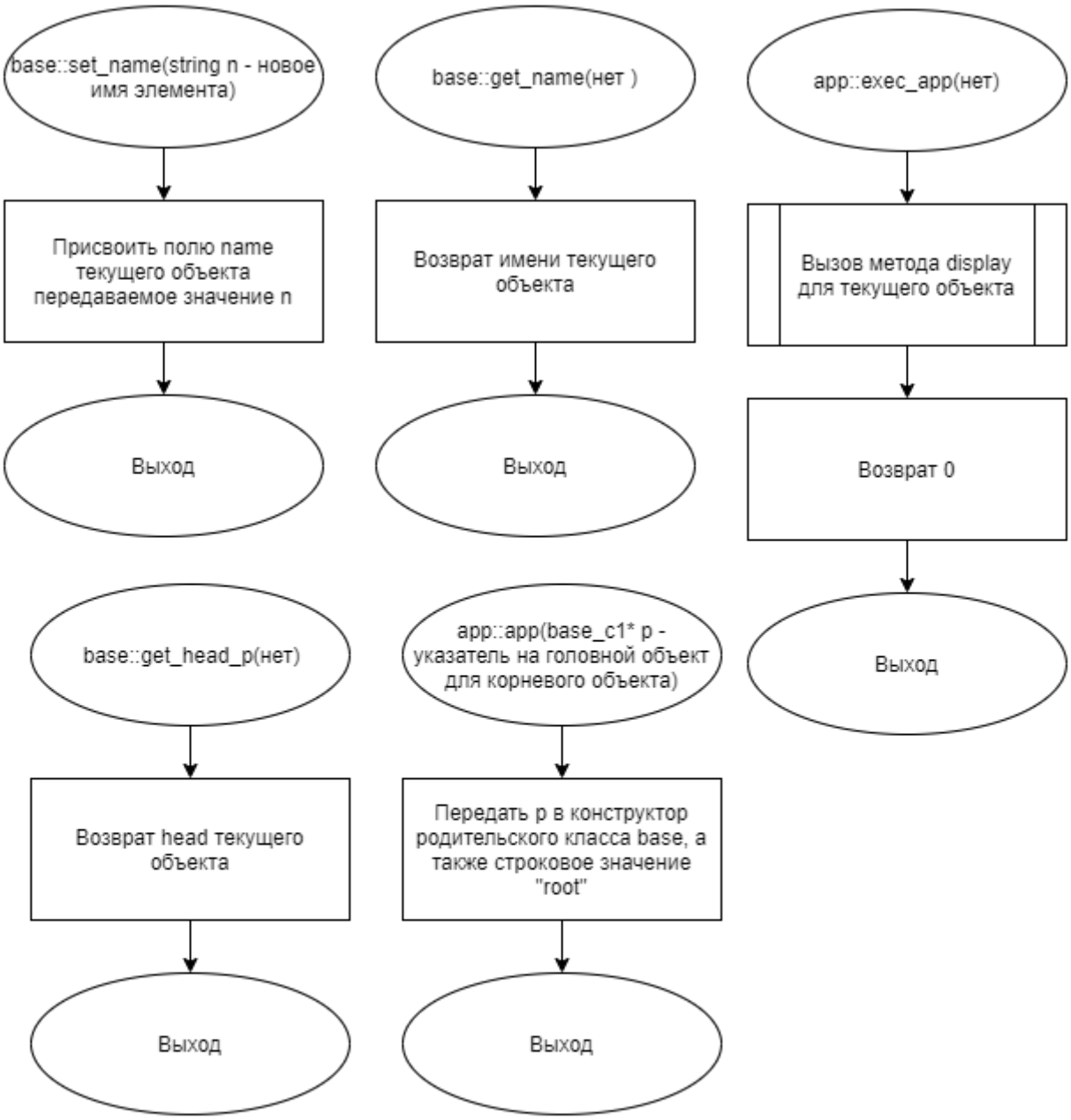

Рис. 2. Блок-схема алгоритма.

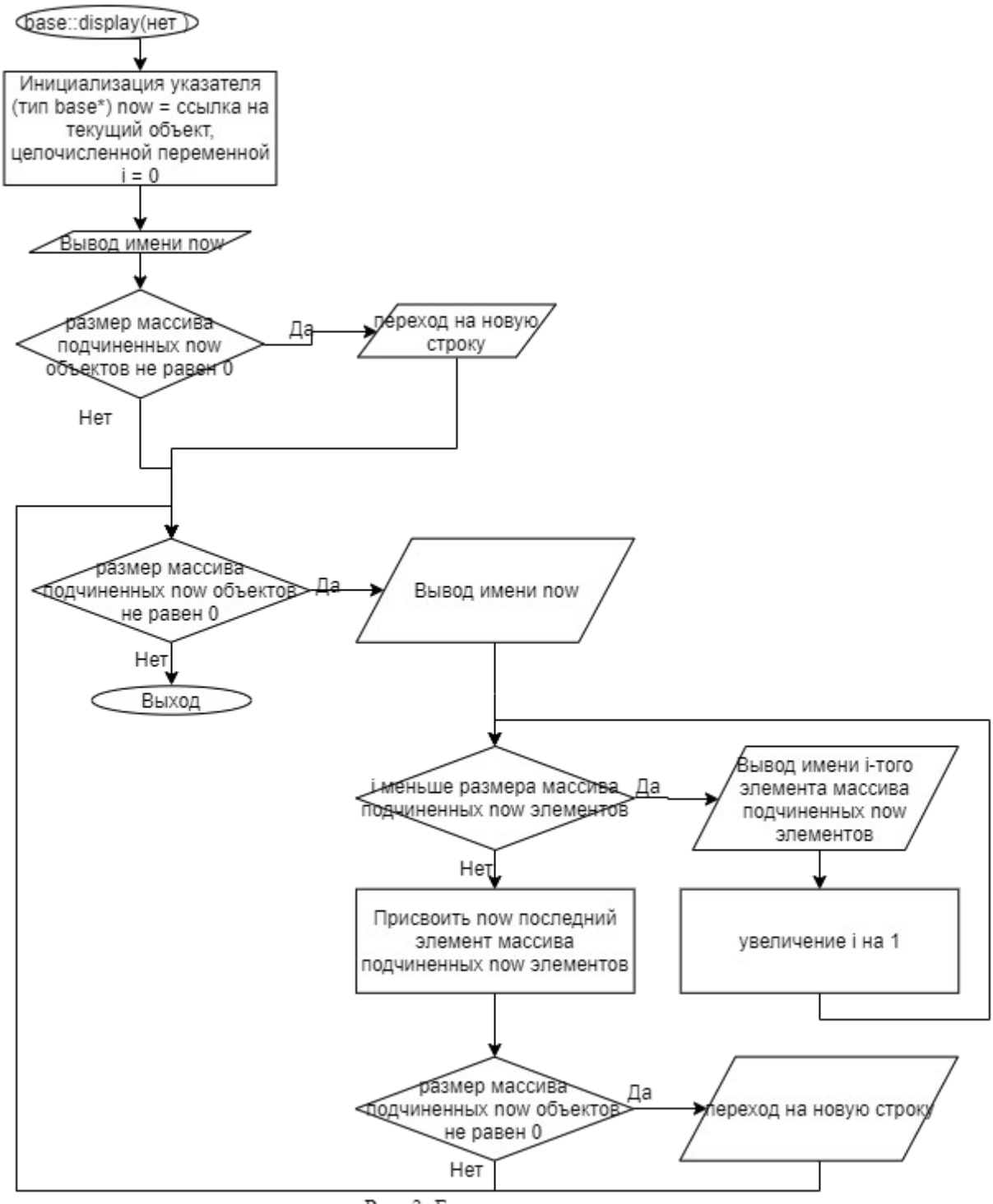

Рис. 3. Блок-схема алгоритма.

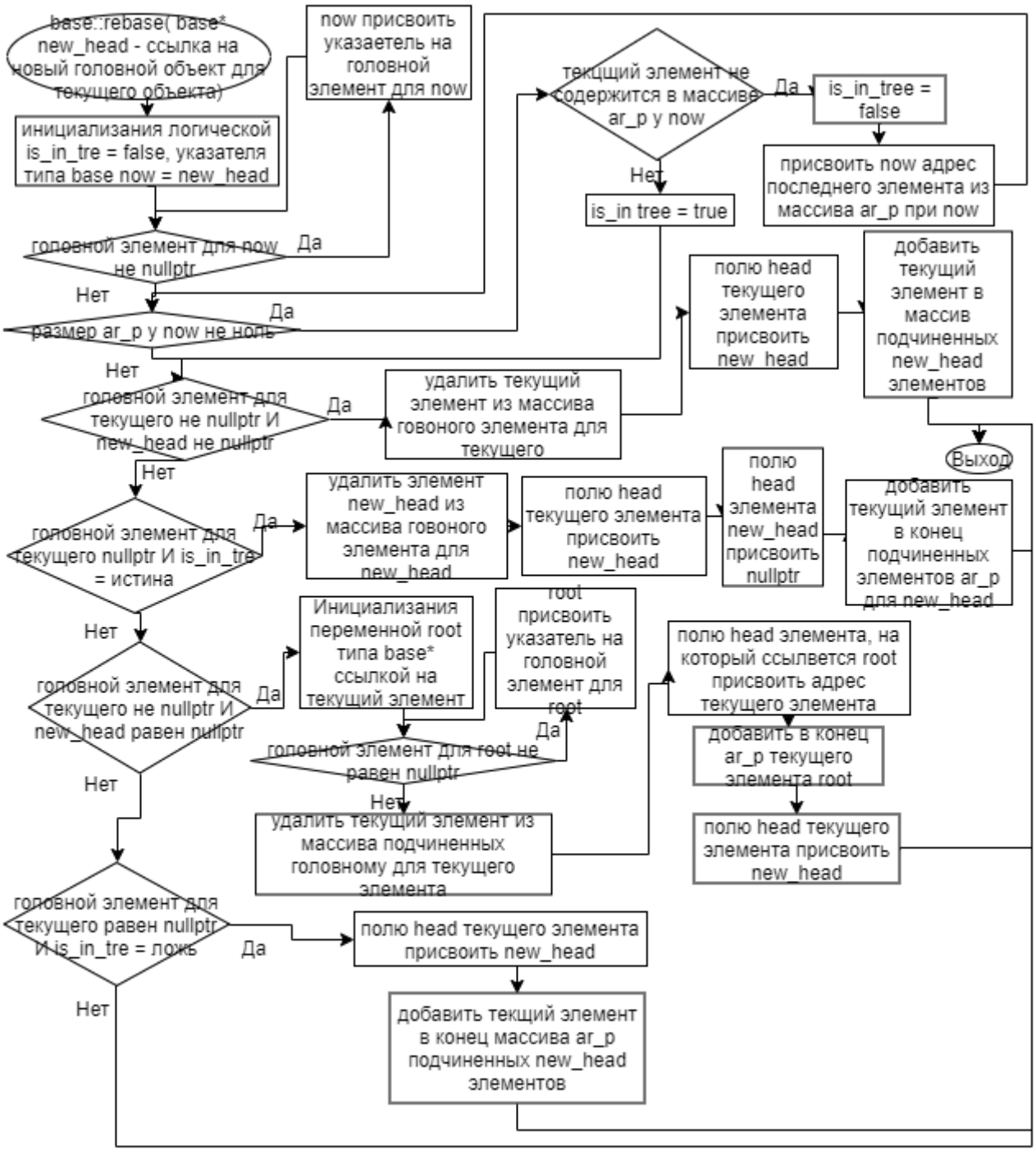

Рис. 4. Блок-схема алгоритма.

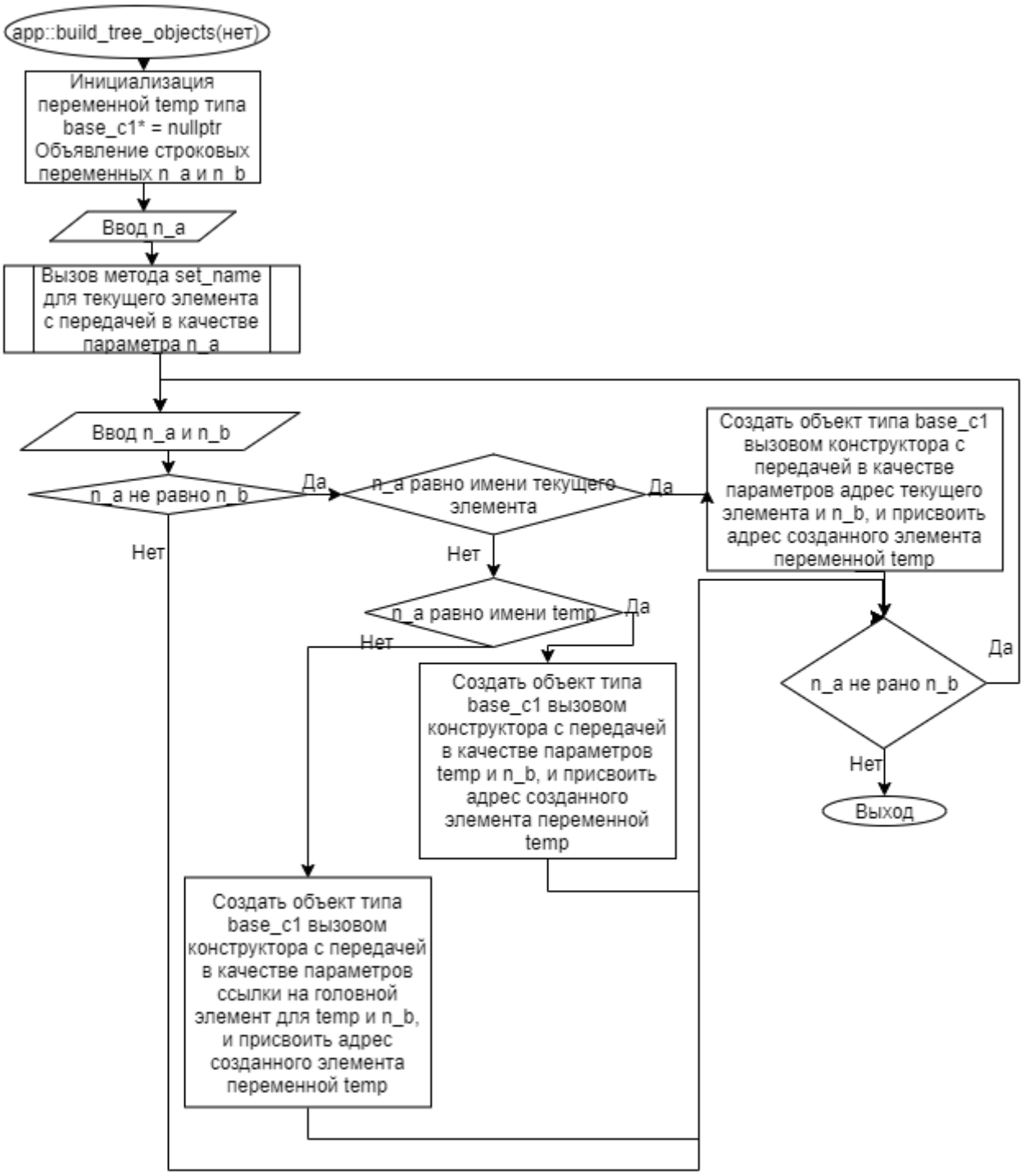

Рис. 5. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл app.cpp

```
#include "app.h"
#include <iostream>
#include <string>

app::app(base_c1* p):base(p, "root")
{
}

void app::build_tree_objects()
{
        base_c1* temp = nullptr;
        std::string n_a, n_b;
        std::cin >> n_a;
        this->set_name(n_a);
        do{
                std::cin >> n_a >> n_b;
                if(n_a!=n_b) {
                        if (n_a == this->get_name()) {
                                temp = new base_c1(this, n_b);
                        }
                        else {
                                if(temp->get_name()==n_a)
                                        temp = new base_c1(temp, n_b);
                                else
                                        temp = new base_c1(temp->get_head_p(),
n_b);
                        }
                }
        } while (n_a != n_b);

}

int app::exec_app()
{
        this->display();
        return 0;
}
```

Файл app.h

```
#ifndef APP_H
#define APP_H
#include<iostream>
```

```
#include "base.h"
#include "base_c1.h"
class app : public base
{
public:
        app(base_c1 *p);
        void build_tree_objects();
        int exec_app();
};
#endif
```

Файл base_c1.cpp

```
#include "base_c1.h"
base_c1::base_c1(base* b, string n) :base(b, n) {}
```

Файл base_c1.h

```
#ifndef BASE_C1_H
#define BASE_C1_H
#include "base.h"
using namespace std;
class base_c1 :
        public base
{
public:
        base_c1(base* b, string n);
};

#endif
```

Файл base.cpp

```
#include "base.h"
#include <iostream>
#include <algorithm>
using namespace std;
base::base(base* h, std::string n)
{
        this->head = h;
        name = n;
        if (this->head != nullptr) {
                this->head->ar_p.emplace(this->head->ar_p.end(),this);
```

```
        }
}

void base::set_name(std::string n)
{
        this->name = n;
}

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

void base::display()
{
        base* now = this;
        while (now->get_head_p() != nullptr) {
                now = now->get_head_p();
        }
        std::cout << now->get_name();
        if (now->ar_p.size() != 0) cout << endl;
        while (now->ar_p.size() != 0) {
                std::cout << now->name;
                for (int i = 0; i < now->ar_p.size(); i++) {
                        std::cout<< "  " << now->ar_p[i]->get_name() ;
                }
                now = now->ar_p[now->ar_p.size() - 1];
                if(now->ar_p.size() != 0)
                        std::cout << std::endl;
        }
}

void base::rebase(base* new_head)
{
        bool is_in_tre = false;

        base* now = new_head;
        while (now->get_head_p() != nullptr) {
                now = now->get_head_p();
        }
        while (now->ar_p.size() != 0) {
                if (find(now->ar_p.begin(), now->ar_p.end(), this) == now-
>ar_p.end())
                        is_in_tre = false;
                else {
                        is_in_tre = true;
                        break;
                }
                now = now->ar_p[now->ar_p.size() - 1];
        }

        if (this->get_head_p() != nullptr && new_head != nullptr) {
                this->get_head_p()->ar_p.erase(find(this->get_head_p()-
>ar_p.begin(), this->get_head_p()->ar_p.end(),this));
                this->head = new_head;
                new_head->ar_p.emplace(new_head->ar_p.begin(),this);
        }
        else if (this->get_head_p() == nullptr && is_in_tre) {
                new_head->get_head_p()->ar_p.erase(find(new_head-
>get_head_p()->ar_p.begin(), new_head->get_head_p()->ar_p.end(), new_head));
```

```
                this->head = new_head;
                new_head->head = nullptr;
                new_head->ar_p.emplace(new_head->ar_p.begin(), this);
        }
        else if (this->get_head_p() != nullptr && new_head == nullptr) {
                base* root = this;
                while (root->get_head_p() != nullptr) {
                        root = root->get_head_p();
                }
                this->get_head_p()->ar_p.erase(find(this->get_head_p()-
>ar_p.begin(), this->get_head_p()->ar_p.end(), this));
                root->head = this;
                this->ar_p.emplace(this->ar_p.end(), root);
                this->head = new_head;
        }
        else if((this->get_head_p() == nullptr && !is_in_tre)){
                this->head = new_head;
                new_head->ar_p.emplace(new_head->ar_p.end(), this);
        }
}


base* base::get_head_p()
{
        return this->head;
}
```

Файл base.h

```
#ifndef BASE_H
#define BASE_H
#include <string>
#include <vector>
class base
{
protected:
        std::string name;
        base* head;
        //base* root;
        std::vector<base*> ar_p;
public:

        base(base*, std::string="Def_name");
        void set_name(std::string);
        std::string get_name();
        void display();
        void rebase(base* new_per);
        base* get_head_p();

};
```

```
#endif
```

Файл main.cpp

```
#include "app.h"
int main()
{
        app q(nullptr);
        q.build_tree_objects();
        return q.exec_app();
}
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
o1 o0 o0	o1	o1
o1 o1 o2 o1 o3 o1 o4 o4 o5 o4 o6 o0 o0	o1 o1 o2 o3 o4 o4 o5 o6	o1 o1 o2 o3 o4 o4 o5 o6
o1 o1 o2 o0 o0	o1 o1 o2	o1 o1 o2
o1 o1 o2 o2 o3 o3 o4 o0 o0	o1 o1 o2 o2 o3 o3 o4	o1 o1 o2 o2 o3 o3 o4

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (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_laboratornyh_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).