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-27-21	Шевелёв И.А.
Руководитель практики	Ассистент	Морозов В.А.
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

Иерархическое построение объектов

Класс string библиотеки <string>

Класс vector библиотеки <vector>

Объекты ввода/вывода потока данных (cout/cin библиотеки <iostream>)

Основная программа:

Объект класса Application

Метод build объекта obj

Метод launch

Объект obj класса Application

Класс Tree

• Поля

Модификатор доступа protected

string name - Строка с именем объекта

Tree* head - указатель на родительский объект

vector<Tree*> children - динамический массив, хранящий в себе объекты и прямых его наследников

Модификатор доступа public

Условный оператор if

Ключевое слово nullptr

Указатель

Цикл for

Оператор возрата return

Методы:

Tree(Tree* head, string name = "root") - Параметризированный конструктор

void set_name(string name) - Задаёт имя объекту

string get_name() - Возращает имя объекта

void set_head(Tree* head) - Переопределяет головной объект

Tree* get_head() - Возращает головной объект

void print() - Печатает дерево объектов

Tree* getByName(string name) - Поиск объекта по имени

Производный класс Application наследуемого класса Tree

• Поля

Модификатор доступа public

Цикл while

Условный оператор if

Указатель

Методы:

Application(Tree* head) - Конструктор класса

void build() - Метод построение дерева объектов

int launch() - Запуск системы

Производный класс Class1 наследуемого класса Tree

• Поля

Модификатор доступа public

Методы:

Class1(Tree* head, string name) - Конструктор класса

Производный класс Class2 наследуемого класса Tree

• Поля

Модификатор доступа public

Методы:

Class2(Tree* head, string name) - Конструктор класса

Описание алгоритма

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

Конструктор класса: Tree

Модификатор доступа: public

Функционал: Конструктор

Параметры: Tree* head, string name. Указатель на объект, имя

Алгоритм конструктора представлен в таблице 1.

Таблица 1. Алгоритм конструктора класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Присвоение свойству name текущего объекта значение параметра name	2	
2	head != nullptr		3	
			∅	
3		Присвоение свойству head текущего объекта значение параметра head	4	
4		Добавление текущего объекта в передоваемый объект head в свойство children	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: set_name

Функционал: Задаёт имя объекту

Параметры: string name - имя объекта

Возвращаемое значение: void, не возращает значения

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода set_name класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Присвоение свойству name текущего объекта значение параметра name	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: get_name

Функционал: Возврат имени объекта

Параметры: нет

Возвращаемое значение: string, возращает строку

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода get_name класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат значения name	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: set_head

Функционал: Переопределения головного объекта

Параметры: Tree* pHead, указатель на объект

Возвращаемое значение: void, не возращает значения

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода set_head класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Присвоение свойству name текущего объекта значение параметра head	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: get_head

Функционал: Возрат текущего головного объекта

Параметры: нет

Возвращаемое значение: Tree*, указатель на объект

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода get_head класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Возрат значение head	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: print

Функционал: Вывод имён объектов

Параметры: нет

Возвращаемое значение: void, не возращает значения

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода print класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1	children.empty()		2	Если вектор пуст
		Возрат без значения	∅	Иначе
2		Вывод перехода на новую строку, name	3	
3		Вывод имени каждого объекта в массиве children	4	
4		Вызов метода print у каждого объекта в массиве children	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: getByName

Функционал: Поиск объекта по имени и его возрат

Параметры: string name, строка имени объекта

Возвращаемое значение: Tree*, указатель на объект

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода getByName класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1	this->name == name	return this	∅	
			2	
2		Объявление указателя result класса Tree	3	
3		Счётчик i с значением	4	
4	i < children.size()	Инкремент i	5	
			7	
5		Присвоение result значение метода getByName(name) объекта из элемента массива children[i]	6	
6	result != nullptr	Возрат result	∅	
			4	
7		Возрат nullptr	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: build

Функционал: Построение дерево объектов

Параметры: нет

Возвращаемое значение: void, не возращает значение

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода build класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление переменных parentName, childName строкового типа данных	2	
2		Ввод значения переменной parentName	3	
3		Вызов метода set_name передаваемого в параметры значения parentName	4	
4	true		5	
			∅	
5		Ввод значения parentName, childName	6	
6	parentName == childName	return	∅	
			7	
7	parentName == this->get_name()	Инцилизация указателя child под выделенную память объекта класса Class1 с параметрами this, childName	4	
			8	
8		Инцилизация указателя child под выделенную память объекта класса Class2 с параметрами this->getByName(parentName), childName	4	

Класс объекта: Application

Модификатор доступа: public

Метод: launch

Функционал: Метод начало работы программы

Параметры: нет

Возвращаемое значение: int, 0

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода launch класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод значения name	2	
2		Вызов метода print	3	
3		return 0	∅	

Конструктор класса: Class1

Модификатор доступа: public

Функционал: Конструктор

Параметры: Tree* head, string name. Указатель на объект, имя объекта

Алгоритм конструктора представлен в таблице 10.

Таблица 10. Алгоритм конструктора класса Class1

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Конструктор класса: Class2

Модификатор доступа: public

Функционал: Конструктор

Параметры: Tree* head, string name. Указатель на объект, имя объекта

Алгоритм конструктора представлен в таблице 11.

Таблица 11. Алгоритм конструктора класса Class2

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Конструктор класса: Application

Модификатор доступа: public

Функционал: Конструктор

Параметры: Tree* head. Указатель на объект

Алгоритм конструктора представлен в таблице 12.

Таблица 12. Алгоритм конструктора класса Application

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Функция: main

Функционал: Главная функция программы

Параметры: нет

Возвращаемое значение: int, Успешное выполнение программы (значение 0)

Алгоритм функции представлен в таблице 13.

Таблица 13. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Создание объекта obj с параметром nullptr	2	
2		Вызов метода build() объекта obj	3	
3		Возврат значение метода launch объекта obj	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

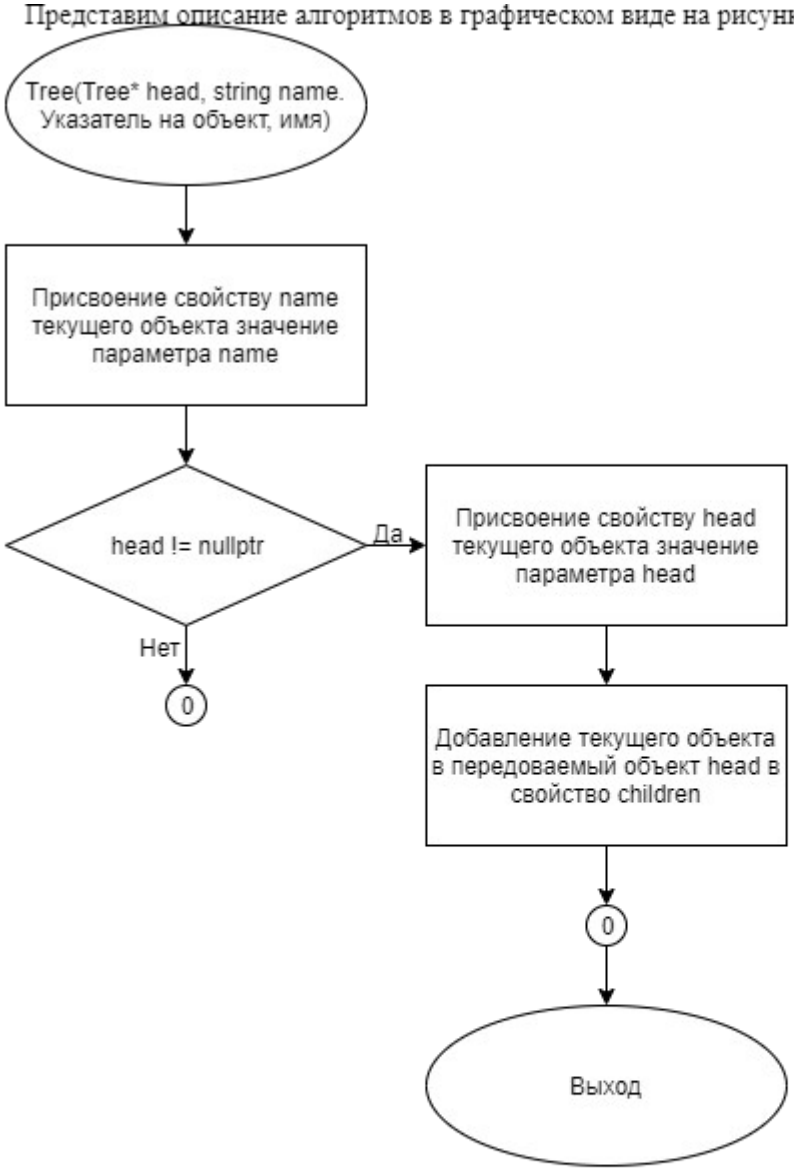

Рис. 1. Блок-схема алгоритма.

Рис. 2. Блок-схема алгоритма.

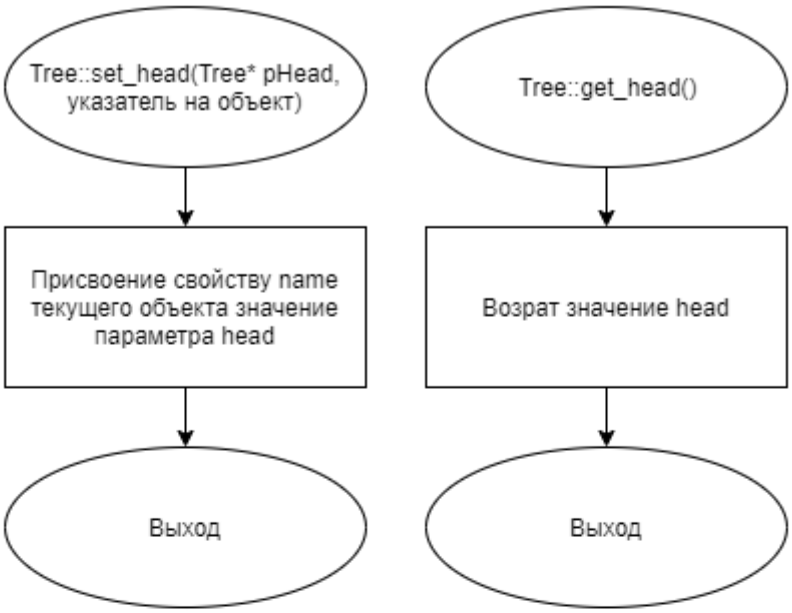

Рис. 3. Блок-схема алгоритма.

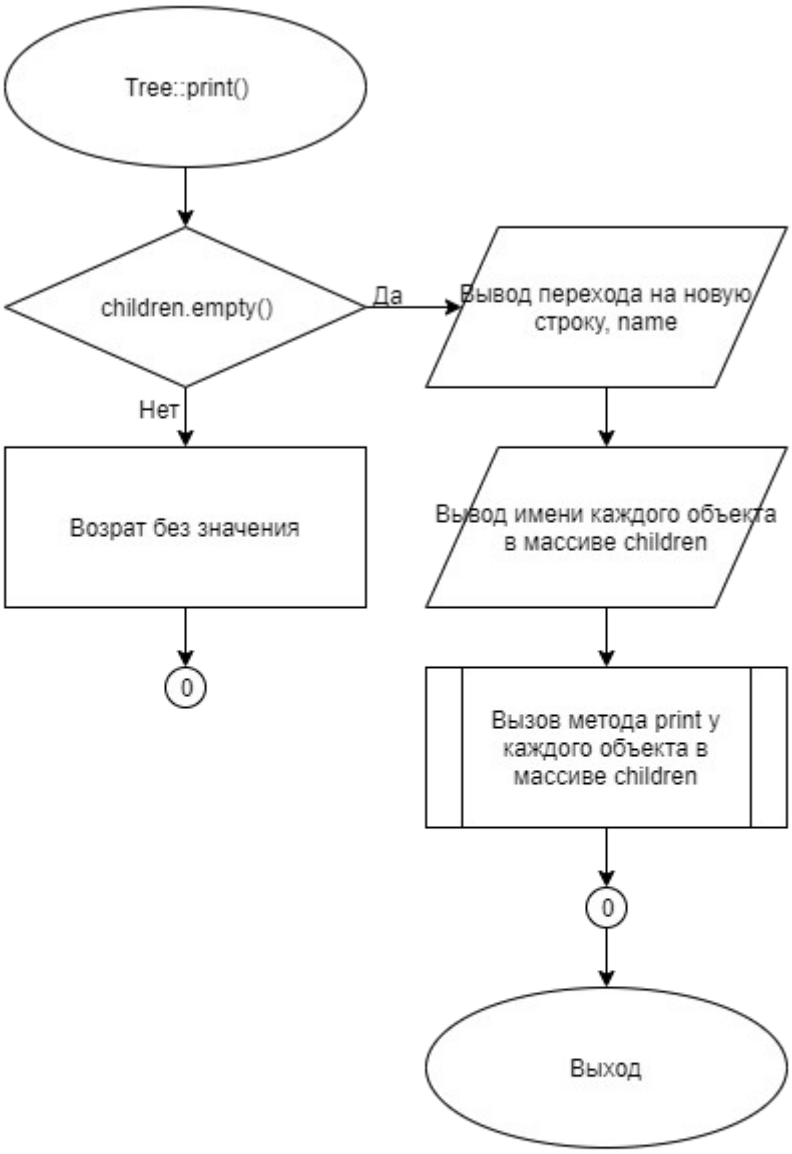

Рис. 4. Блок-схема алгоритма.

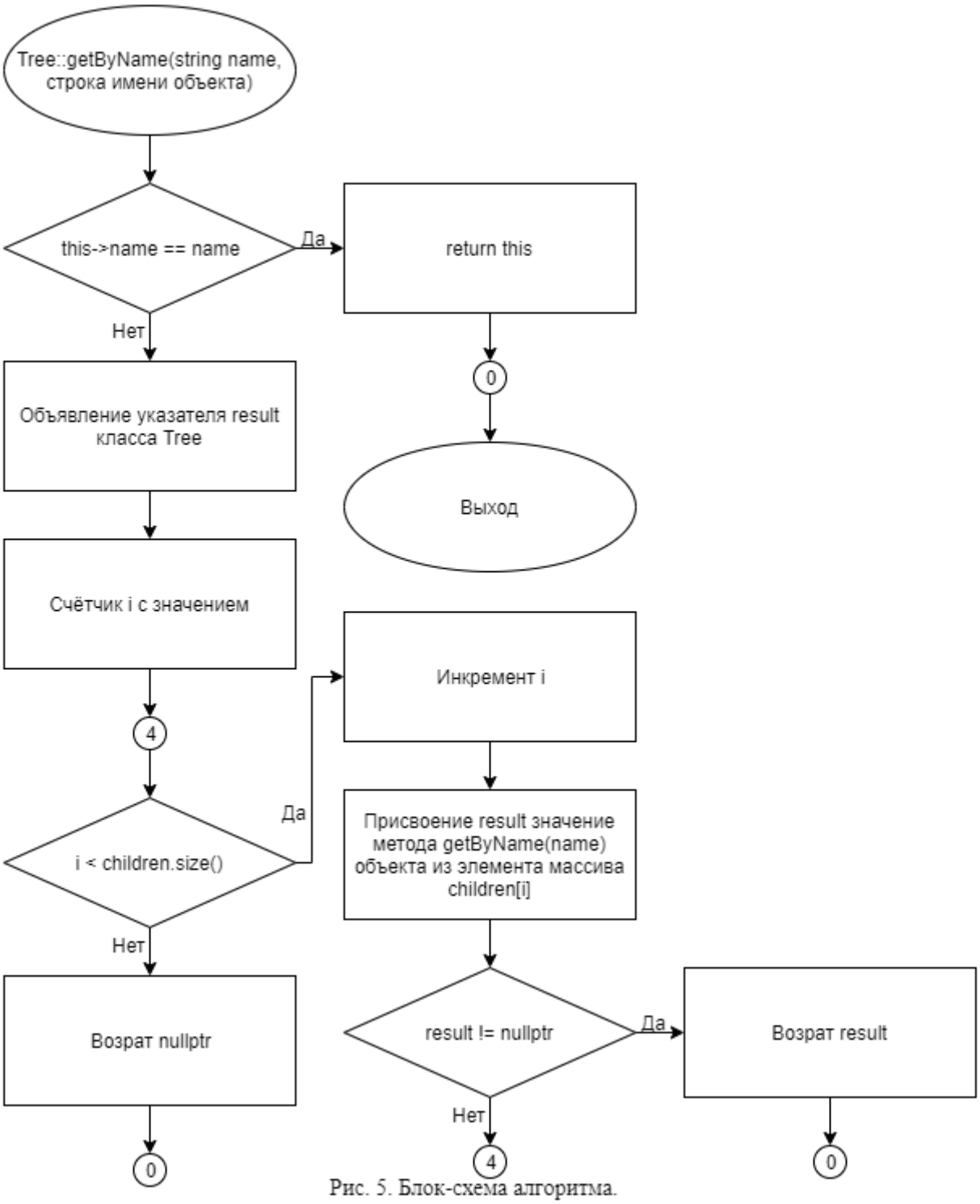

Рис. 5. Блок-схема алгоритма.

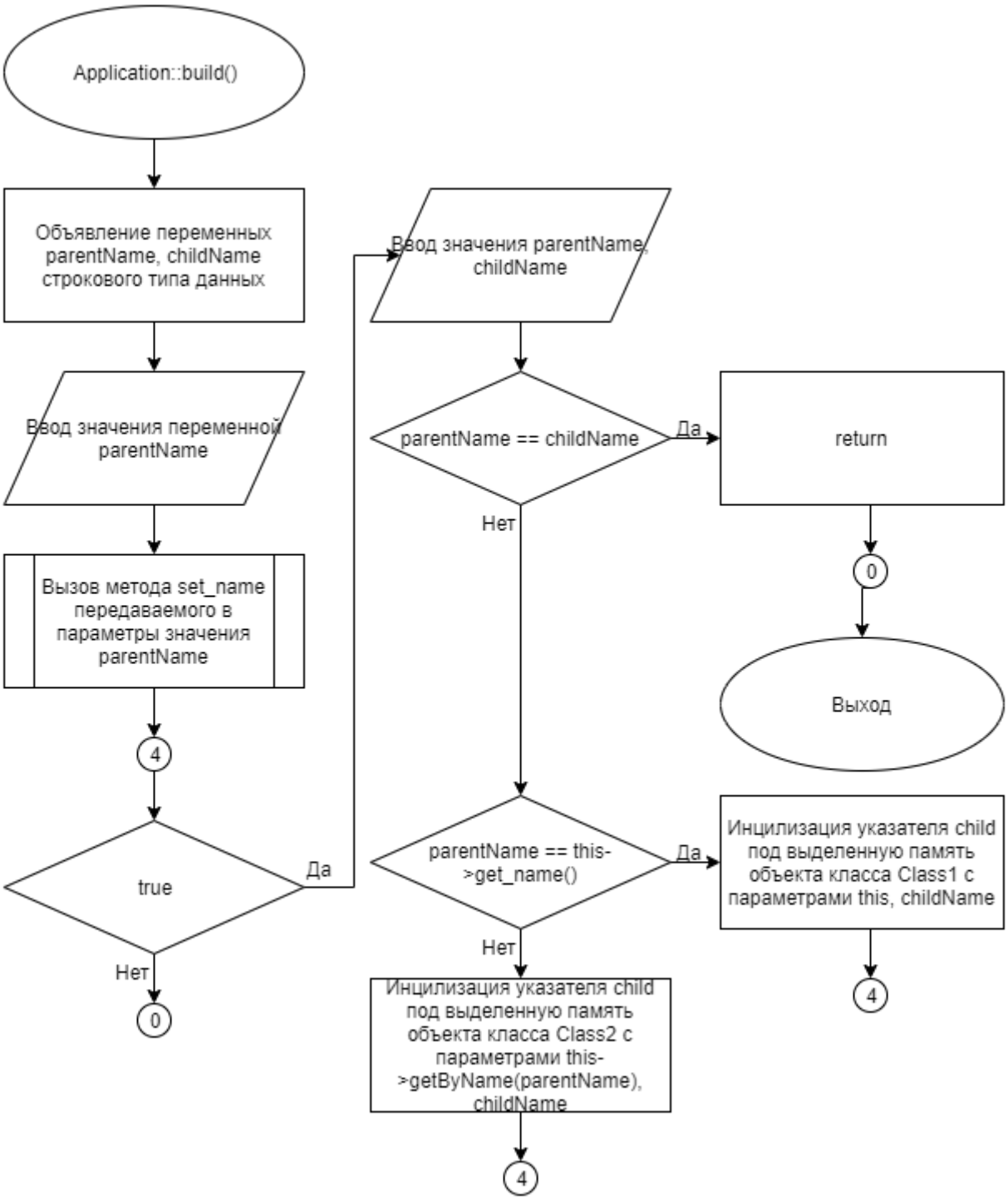

Рис. 6. Блок-схема алгоритма.

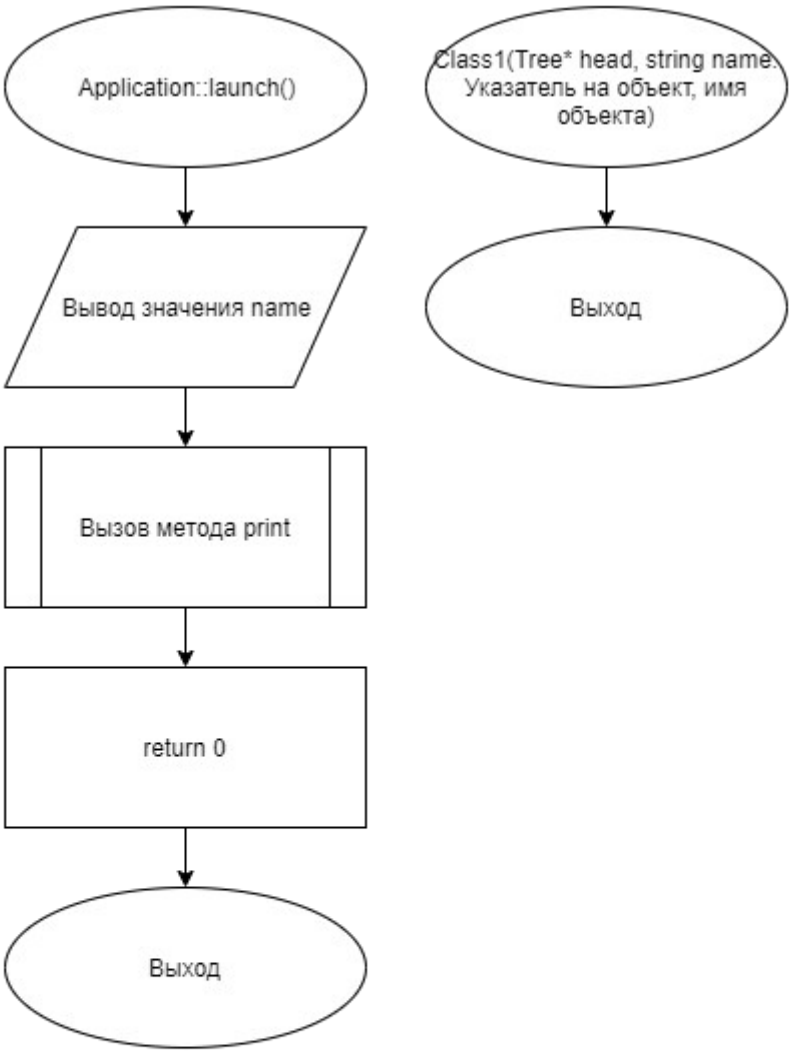

Рис. 7. Блок-схема алгоритма.

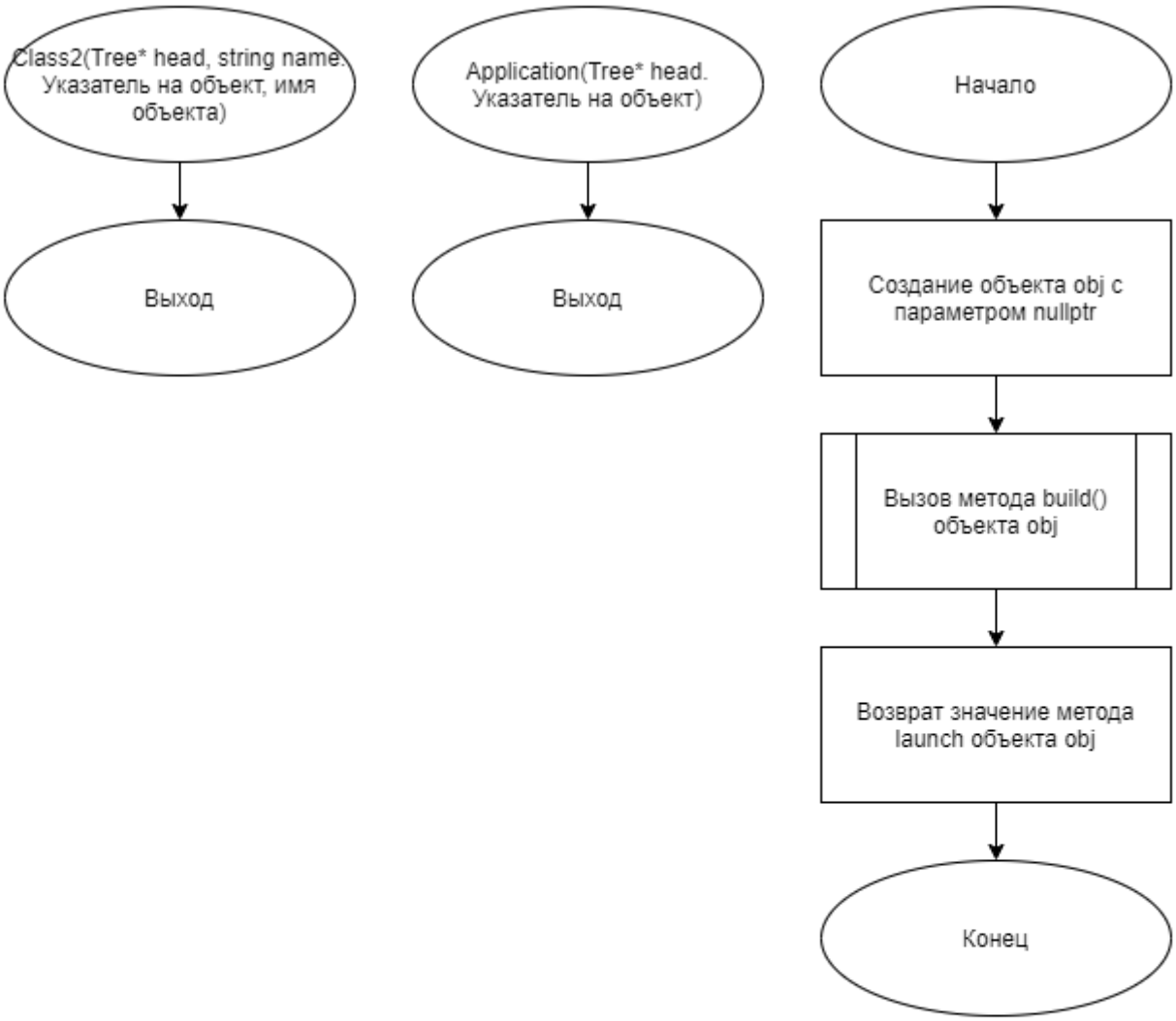

Рис. 8. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Application.cpp

```
#include "Application.h"

Application::Application(Tree* head): Tree(head){}

void Application::build(){
        string parentName, childName;
        cin >> parentName;
        set_name(parentName);
        while(true){
                cin >> parentName >> childName;
                if(parentName == childName){
                        return;
                }
                if(parentName == this->get_name()){
                        Class1* child = new Class1(this, childName);
                }
                else {
                        Class2* child = new Class2(this-
>getByName(parentName), childName);
                }

        }
}
int Application::launch(){
        cout << name;
        print();
        return 0;
}
```

Файл Application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H
#include "Tree.h"
#include "Class1.h"
#include "Class2.h"

class Application : public Tree{
public:
```

```
        Application(Tree* head);
        void build();
        int launch();
};
#endif
```

Файл Class1.h

```
#ifndef CLASS1_H
#define CLASS1_H
#include "Tree.h"

class Class1 : public Tree{
public:
        Class1(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл Class2.h

```
#ifndef CLASS2_H
#define CLASS2_H
#include "Tree.h"

class Class2 : public Tree{
public:
        Class2(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл main.cpp

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

int main()
{
        // program here
        Application obj(nullptr);
        obj.build();
        return obj.launch();

}
```

Файл Tree.cpp

```
#include "Tree.h"

Tree::Tree(Tree* head, string name){
        this->name = name;
        set_name(name);
        if(head != nullptr){
                this->head = head;
                head->children.push_back(this);
        }
}

void Tree::set_name(string name){
        this->name = name;
}
string Tree::get_name(){
        return name;
}
void Tree::set_head(Tree* head){
        this->head = head;
}
Tree* Tree::get_head(){
        return head;
}
void Tree::print(){
        if(children.empty())
                return;
        cout << endl << name;
        for(Tree* element : children)
                cout << "  " << element->get_name();
        for(Tree* element : children)
                element->print();
}
Tree* Tree::getByName(string name){
        if(this->name == name){
                return this;
        }
        Tree* result;
        for(int i = 0; i < children.size(); i++){
                result = children[i]->getByName(name);
                if(result != nullptr){
                        return result;
                }
        }
        return nullptr;
}
```

Файл Tree.h

```
#ifndef TREE_H
#define TREE_H
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Tree{
protected:
        string name;
        Tree* head;
        vector<Tree*> children;
public:
        Tree(Tree* head, string name = "root");
        void set_name(string name);
        string get_name();
        void set_head(Tree* head);
        Tree* get_head();
        void print();
        Tree* getByName(string name);
};
#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

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
World World India World Canada World Russia Russia Moscow Russia Saint_Petersburg stop stop	World World India Canada Russia Russia Moscow Saint_Petersburg	World World India Canada Russia Russia Moscow Saint_Petersburg
root root 1 root 2 root 3 2 2.1 2 2.2 stop stop	root root 1 2 3 2 2.1 2.2	root root 1 2 3 2 2.1 2.2

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).