МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« КЛ_3_1 Вывод иерархического дерева »

С тудент группы	ИКБО-27-21	Шевелёв И.А.
Руководитель практики	Ассистент	Морозов В.А.

Работа представлена «___»_____2021 г. _____

(подпись студента)

Оценка _____

(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Постановка задачи

Формирование и работа с иерархией объектов программы-системы.

Создание объектов и построение исходного иерархического дерева объектов. Система собирается из объектов, принадлежащих определенным классам. В тексте постановки задачи классу соответствует уникальный номер. Относительно номера класса определяются требования (свойства, функциональность).

Первоначальная сборка системы (дерева иерархии объектов, программы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Корневой объект объявляется в основной программе (main).

Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта»«Наименование очередного объекта»«Номер класса принадлежности очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных программах динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:
- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr);
- метод вывода дерева иерархии объектов;
- метод вывода дерева иерархии объектов и отметок их готовности;
- метод установки готовности объекта реализовать (доработать) следующим образом.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется.
При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:
1. Построить дерево объектов системы (в методе коневого объекта построения исходного дерева объектов).
2. В методе корневого объекта запуска моделируемой системы реализовать:
2.1. Вывод на консоль иерархического дерева объектов в следующем виде:

root

ob_1

ob_2

ob_3

ob_4

ob_5

ob_6

ob_7

где: root - наименование корневого объекта (приложения).

2.2. Переключение готовности объектов согласно входным данным (командам).

2.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

root is ready

ob_1 is ready

ob_2 is ready

ob_3 is ready

ob_4 is not ready

ob_5 is not ready

ob_6 is ready

ob_7 is not ready

Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Последовательность ввода организовано так, что головной объект для

очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка

«Наименование корневого объекта»

Со второй строки

«Наименование головного объекта»«Наименование очередного объекта»«Номер класса принадлежности очередного объекта»

. . . . .

endtree

Со следующей строки вводятся команды включения или отключения объектов

«Наименование объекта»«Номер состояния объекта»

. . . . .

Признаком завершения ввода является конец потока входных данных.

Пример ввода

```
app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
```

```
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
```

Описание выходных данных

Вывести иерархию объектов в следующем виде:

```
Object tree
«Наименование корневого объекта»
 «Наименование объекта 1»
  «Наименование объекта 2»
 «Наименование объекта 3»
. . . . .
The tree of objects and their readiness
«Наименование корневого объекта»«Отметка готовности»
 «Наименование объекта 1»«Отметка готовности»
  «Наименование объекта 2»«Отметка готовности»
 «Наименование объекта 3»«Отметка готовности»
. . . . .
```

«Отметка готовности» - равно «is ready» или «is not ready»
Отступ каждого уровня иерархии 4 позиции.

Пример вывода

```
Object tree
```

app_root

object_01

object_07

object_02

object_04

object_05

The tree of objects and their readiness

app_root is ready

object_01 is ready

object_07 is not ready

object_02 is ready

object_04 is ready

object_05 is not ready

Метод решения

Класс class2, class3, class4, class5, class6 (Наследуют класс Tree)

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

Класс string библиотеки <string>

Шаблоный класс vector библиотеки <vector>

Класс Application (Наследует класс Tree):

Модификатор доступа public

Объект child классов class2, class3, class4, class5, class6

Объекты parentName, childName класса string

Метод print - выводит дерево объектов

Метод launch - запускает программу

Класс Tree:

Модификатор доступа public, protected

Методы set_ready - задаёт объекту состояние готовности

Метод build_ready - задаёт объектами состояние готовности через метод set_ready

Метод print_ready - выводит объекты и их состояние готовности

Метод getByName - поиск объекта по имени

Описание алгоритма

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

Класс объекта: Tree

Модификатор доступа: public

Метод: getByName

Функционал: Поиск объекта по имени

Параметры: string name

Возвращаемое значение: Tree result, nullptr. Возращает объект или нулевой указатель

Алгоритм метода представлен в таблице 1.

Таблица 1. Алгоритм метода getByName класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1	this->name == name	Вернуть текущий объект	∅	
			2	
2		Объявление указателя result класса Tree	3	
3	i < children.size()	Инкремент i	4	
			6	
4		Происвоение result значение метода children[i]->getByName	5	

5	result != nullptr	Вернуть result	∅	
			3	
6	result != nullptr		7	
			9	
7		Свойству result присвоить значение head	8	
8		Вызов метода getByName	6	
9		Вернуть нулевой указатель	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: set_ready

Функционал: Задаёт готовность объекта

Параметры: int input_ready

Возвращаемое значение: void, Не возращает значение

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода set_ready класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1	input_ready == 0	Присвоить свойству объекта ready значение 0	2	
			4	
2	i < children.size()	Инкремент i	3	
			∅	
3		Вызов метода set_ready с параметром 0 объекта children[i]	2	
4		Инцилизация указателя temp класс Tree с значение this->head	5	

5	temp != nullptr		6	
			8	
6	temp->ready == 0	Присвоить свойству объекта ready значение 0. Возрат функции без значения	∅	
			7	
7		Присвоить temp значение temp->head	5	
8		Свойству ready текущего объекта присваивается значение input_ready	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: build_ready

Функционал: Задаёт готовность объектам

Параметры: нет

Возвращаемое значение: void, Не возращает значение

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода build_ready класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление name класса string	2	
2		Объявление input_ready целлочисленного типа данных	3	
3	cin >> name >> input_ready		4	
			∅	

4	getByName(name) != nullptr	Вызов метода set_ready с параметром input_ready возвращаемого объекта из метода getByName(name)	3	
			3	

Класс объекта: Tree

Модификатор доступа: public

Метод: print_ready

Функционал: Вывод имён объектов и их готовность

Параметры: string tab

Возвращаемое значение: void, Не возращает значение

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода print_ready класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1	ready > 0 || ready < 0	Вывод значения метода get_name, " is ready"	2	
		Вывод значения метода get_name, " is not ready"	2	
2	children.size != 0	Присвоение tab прибавление значения " "	3	
			∅	
3	Tree* element : children		4	
			∅	
4		Вывод перехода на новую строку и значение tab	5	
5		Вывод метода print_ready(tab) объекта element	3	

Класс объекта: Tree

Модификатор доступа: public

Метод: build

Функционал: Создание дерево объектов

Параметры: string name

Возвращаемое значение: void, Не возращает значение

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода build класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление parentName, childName класса string	2	
2		Объявление object_number целлочисленного типа данных	3	
3		Ввод значения name	4	
4		Ввод значения parentName	5	
5	parentName == "endtree"	Вызов метода build_ready и выход из цикла	∅	
			6	
6		Ввод значения childName, object_number	7	
7	object_number == 2	Создание объекта child класса Class2 с параметрами значения метода getByname и значение childName	4	
	object_number == 3	Создание объекта child класса Class3 с параметрами значения метода getByname и значение	4	
	object_numbe	Создание объекта child класса	4	

	r == 4	Class4 с параметрами значения метода getByname и значение		
	object_number == 5	Создание объекта child класса Class5 с параметрами значения метода getByname и значение	4	
	object_number == 6	Создание объекта child класса Class6 с параметрами значения метода getByname и значение	4	
			4	

Класс объекта: Application

Модификатор доступа: public

Метод: launch

Функционал: Запуск системы

Параметры: нет

Возвращаемое значение: int, Возращает число 0

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода launch класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Object tree\n"	2	
2		Вызов метода print	3	
3		Вывод "\nThe tree of objects and their readiness"	4	
4		Вызов метода print_ready с параметром ""	5	
5		Возрат 0	∅	

Функция: main

Функционал: Главная функция программы

Параметры: нет

Возвращаемое значение: int, Код возрата

Алгоритм функции представлен в таблице 7.

Таблица 7. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление объекта obj с параметром nullptr	2	
2		Вызов метода build объекта obj	3	
3		Возрат значения метода launch объекта obj	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: set_head

Функционал: Меняет родителя объекта

Параметры: Tree* head

Возвращаемое значение: Tree*, указатель на объект

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода set_head класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1	head != nullptr		2	
			4	

2	i < head->children.size()	Инкремент i	3	
			4	
3	head == this	Вызов метода erase свойства head с параметрами head->children.begin()+i	2	
			2	
4		Свойству текущего объекта head присвоить значение head	5	
5	head != nullptr	Вызов метода push_back свойства children объекта head	∅	
			∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

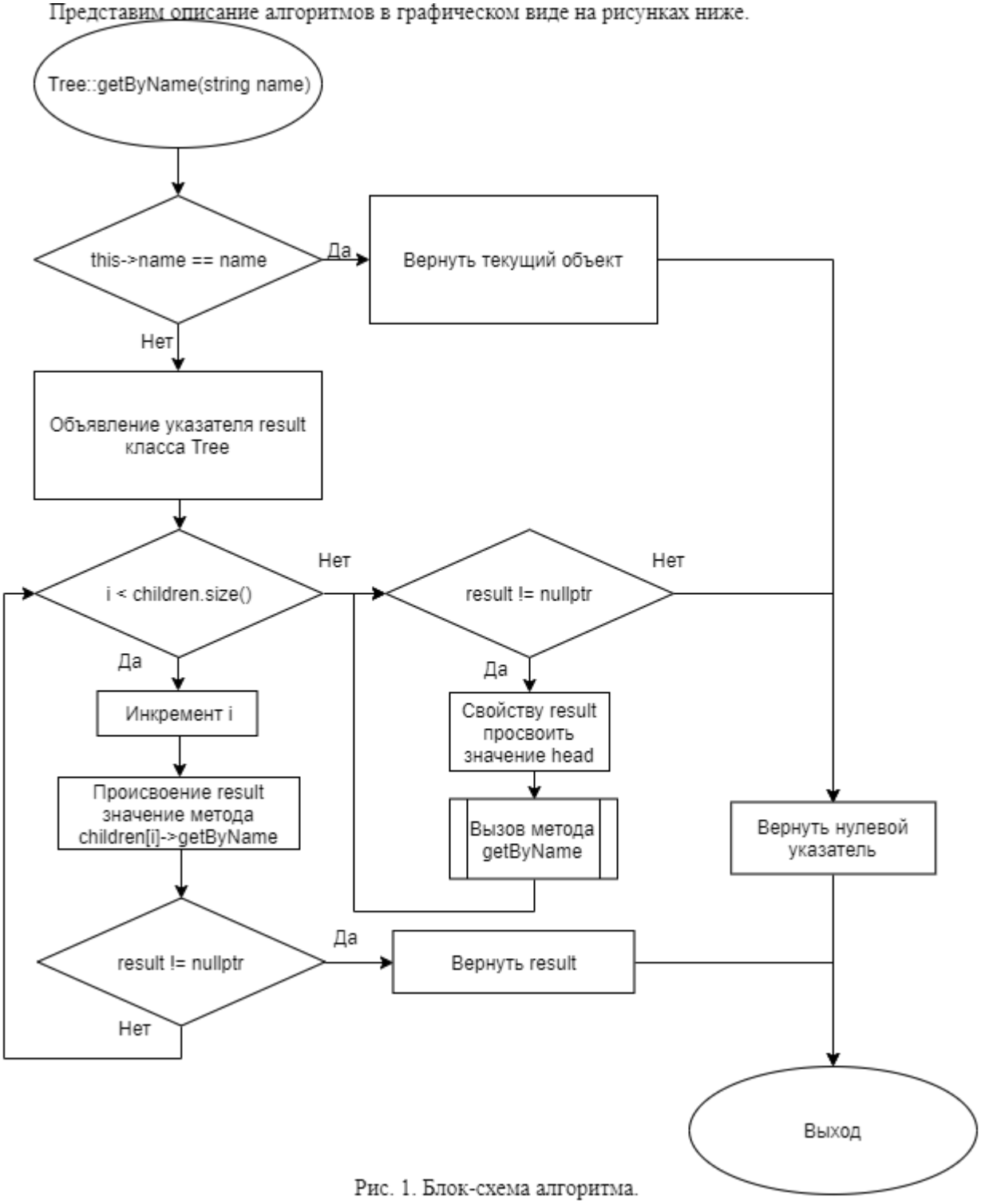

Рис. 1. Блок-схема алгоритма.

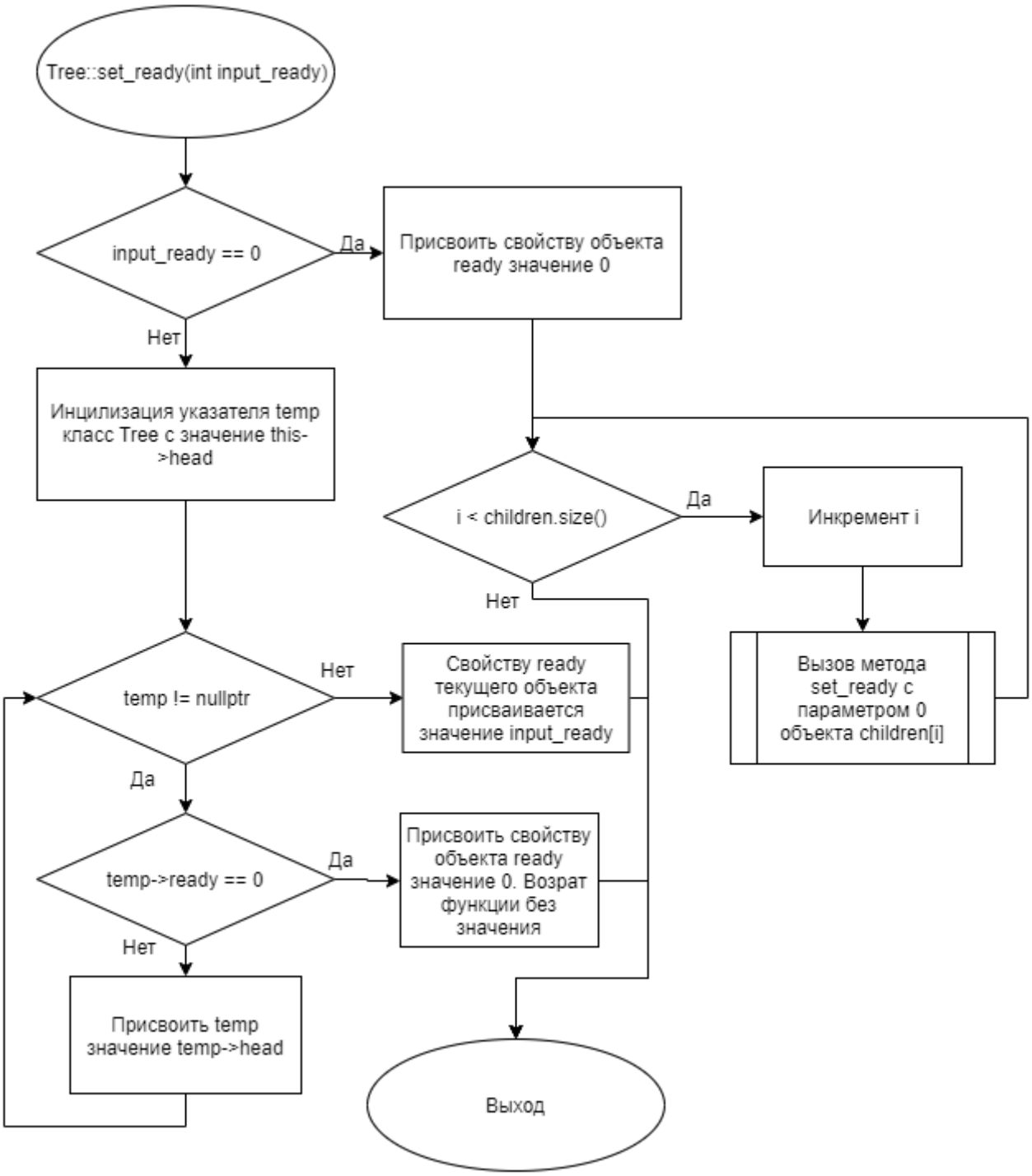

Рис. 2. Блок-схема алгоритма.

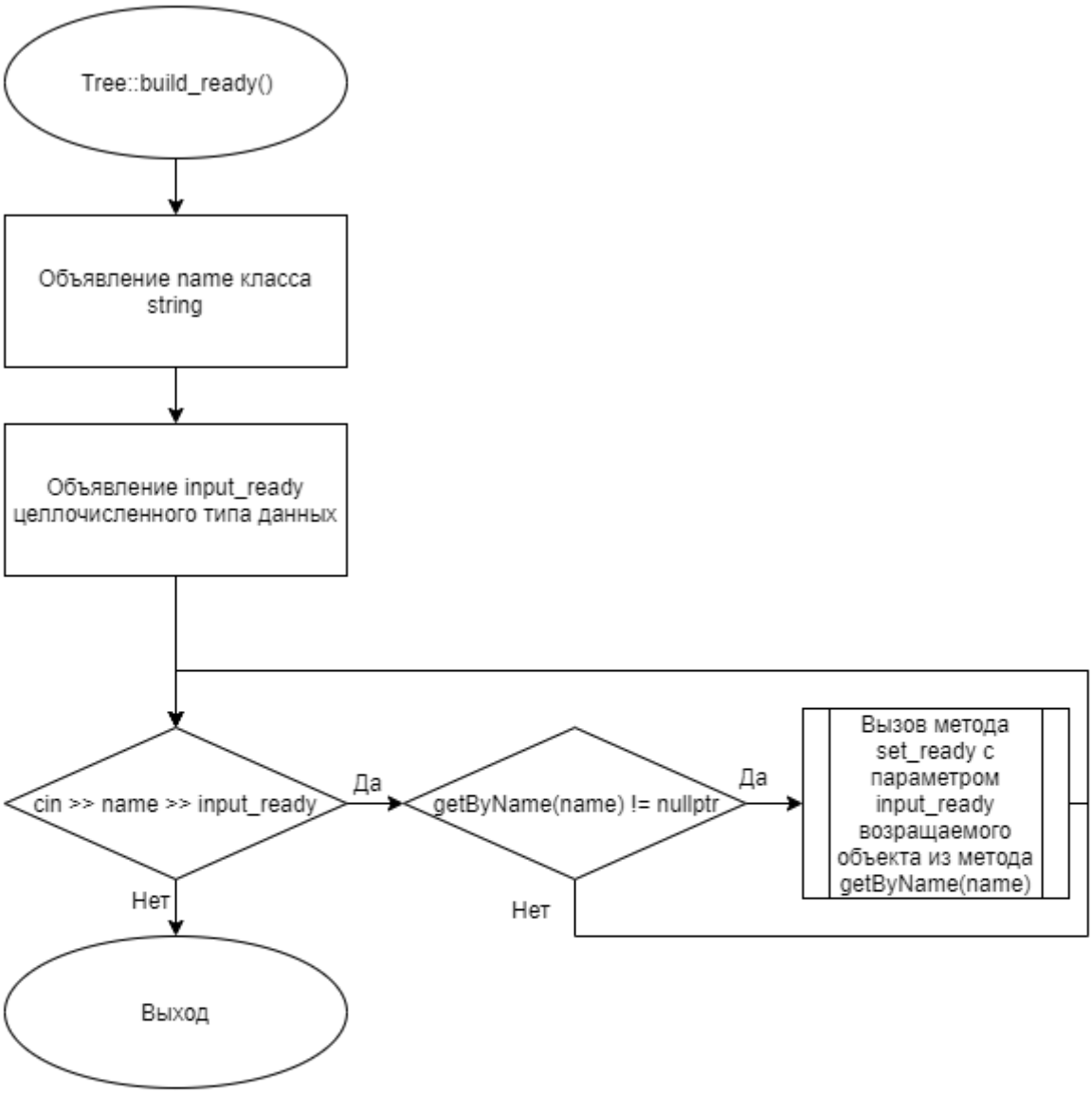

Рис. 3. Блок-схема алгоритма.

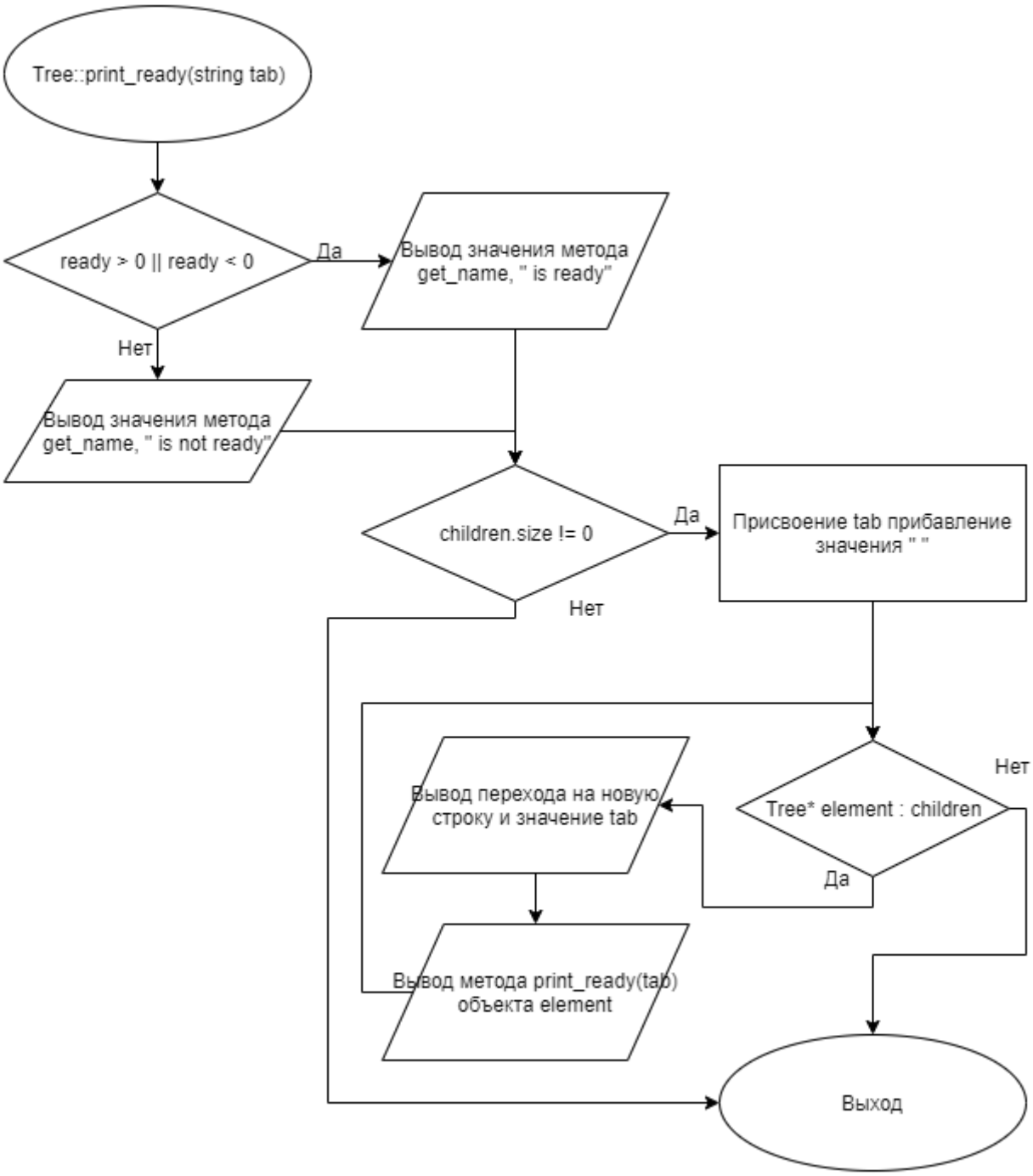

Рис. 4. Блок-схема алгоритма.

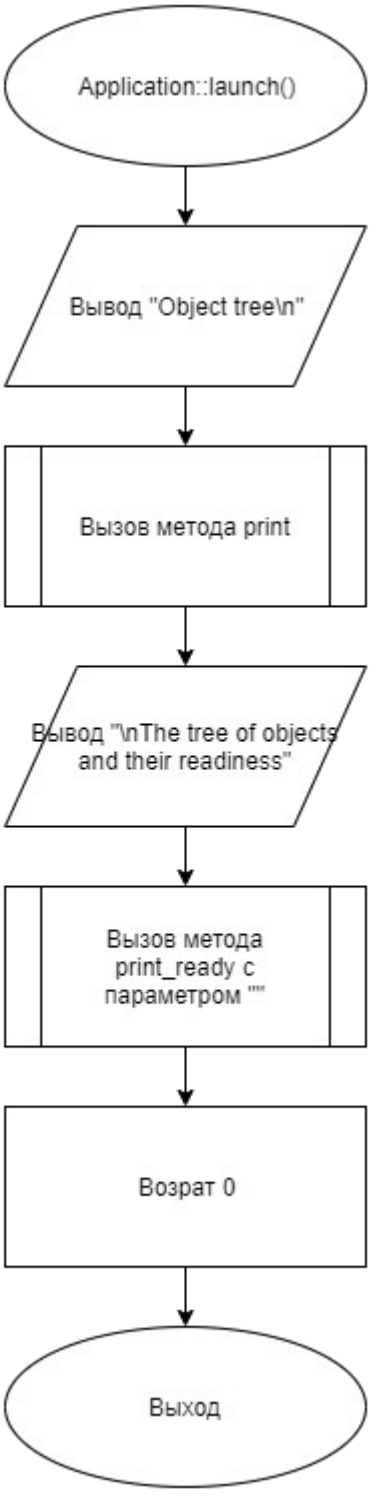

Рис. 5. Блок-схема алгоритма.

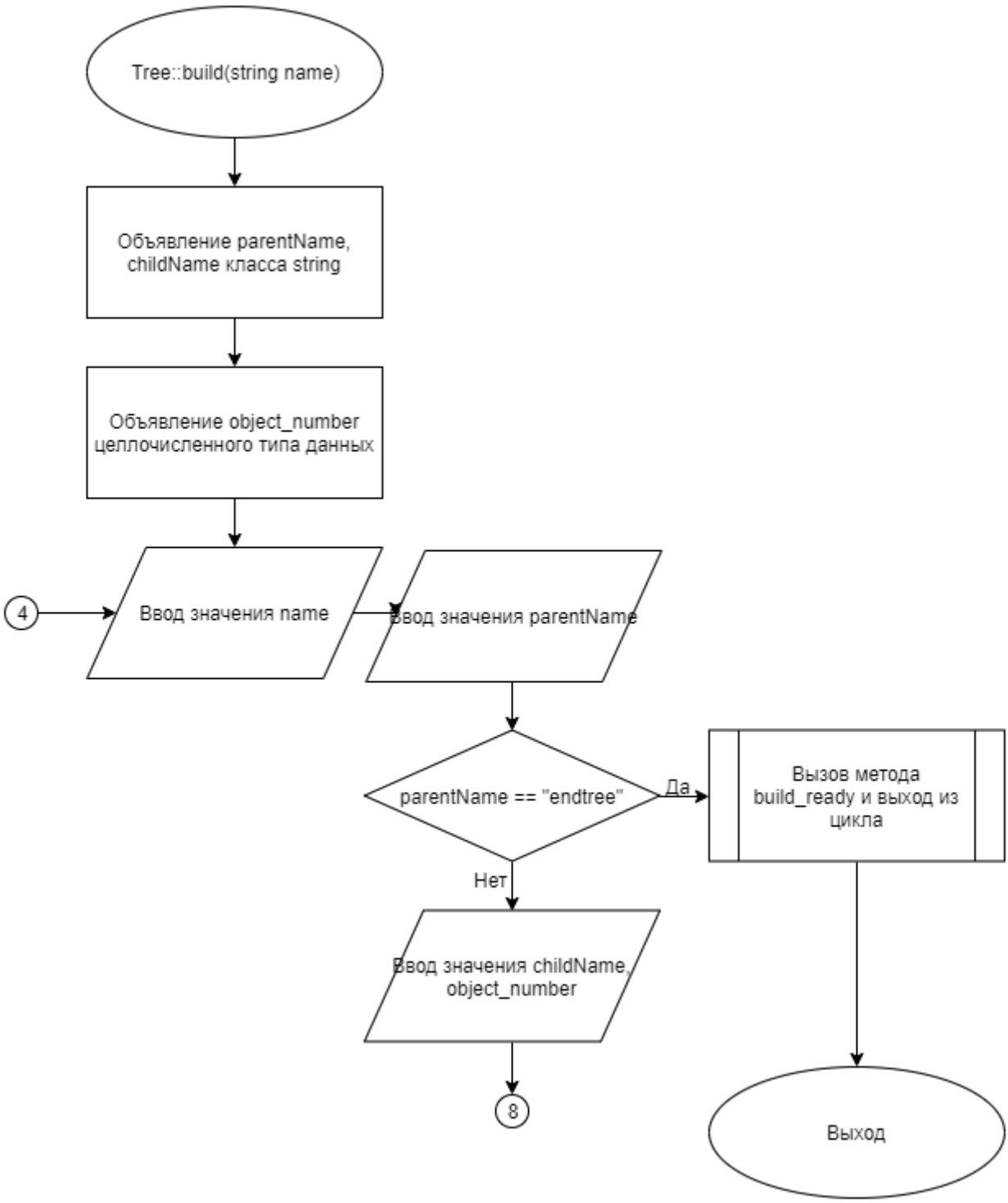

Рис. 6. Блок-схема алгоритма.

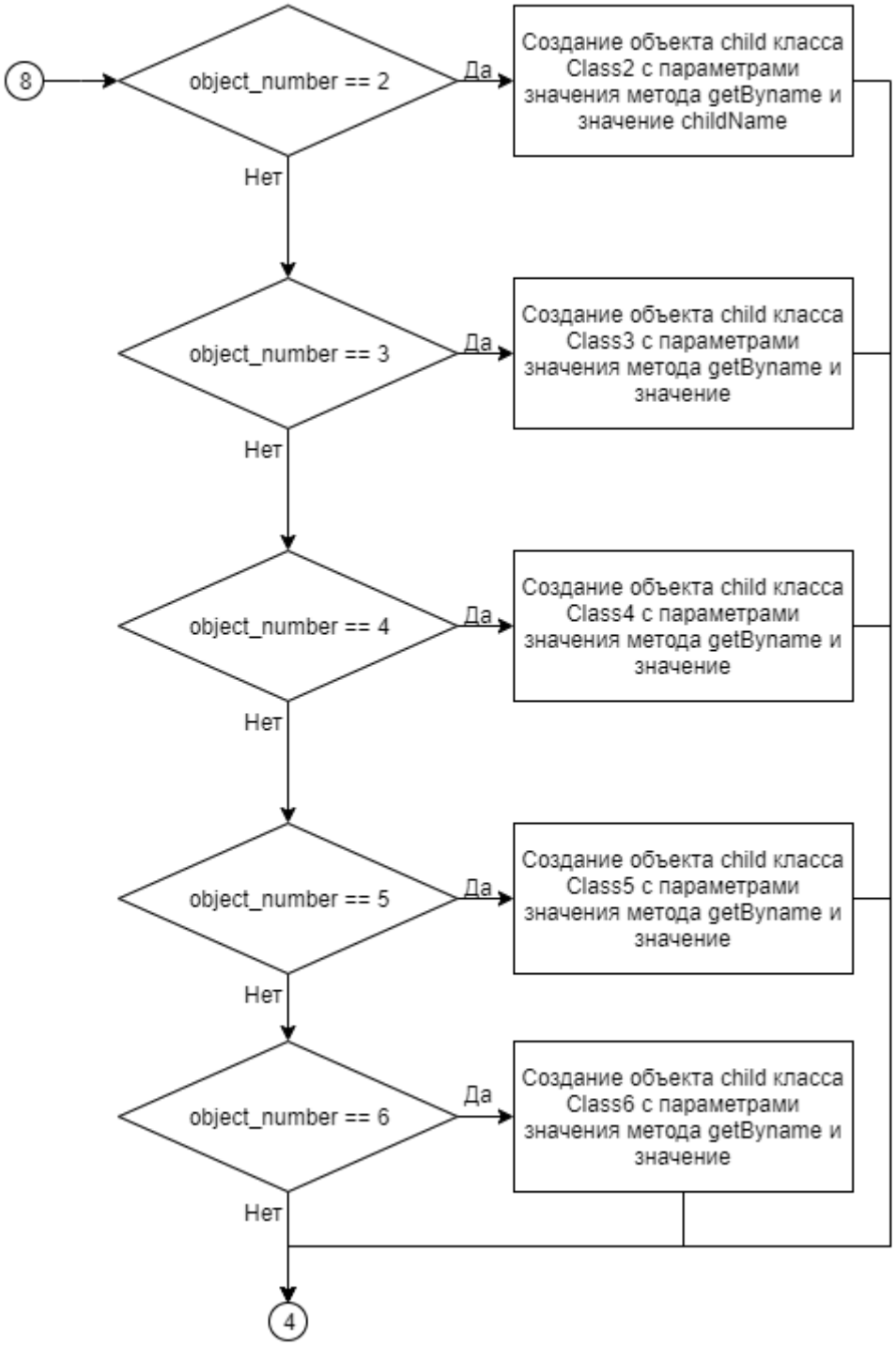

Рис. 7. Блок-схема алгоритма.

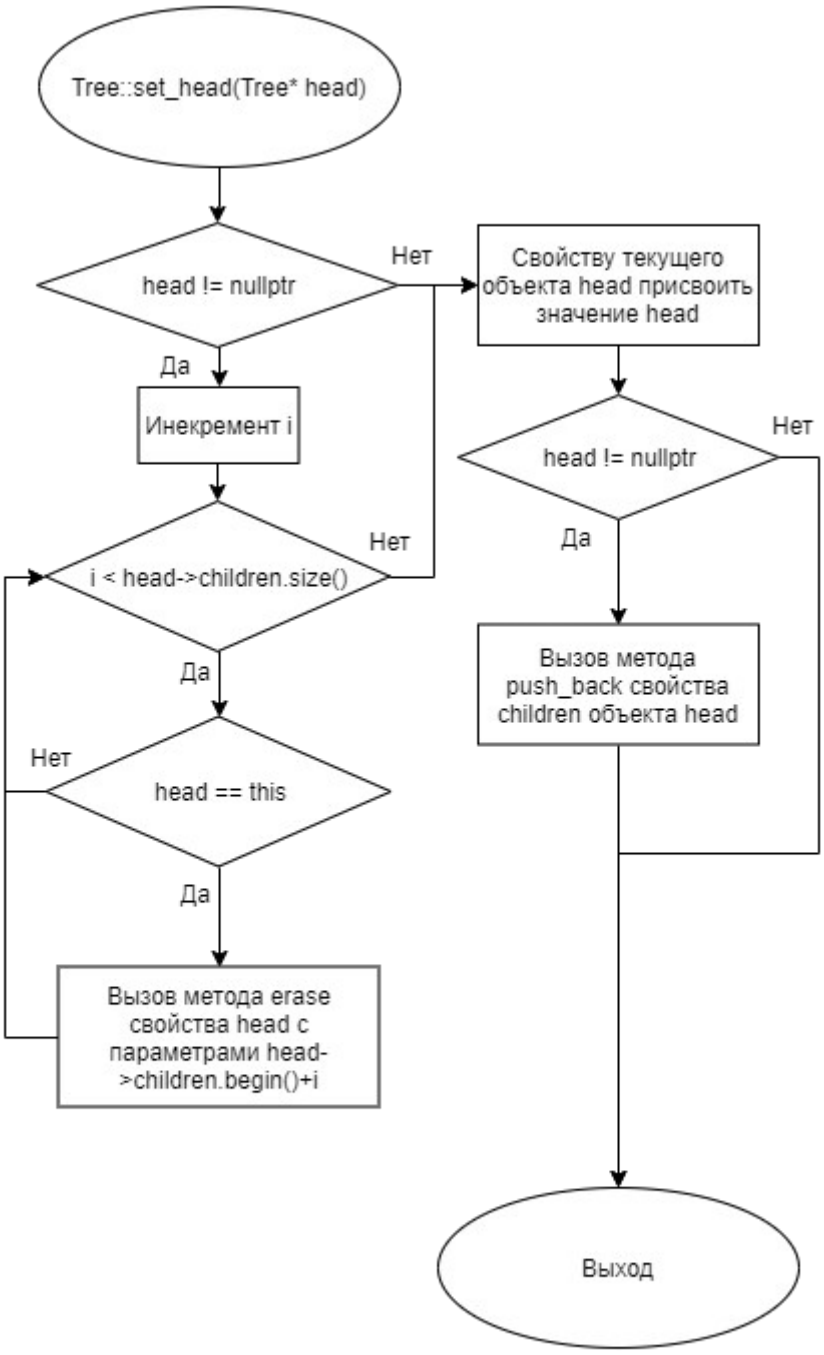

Рис. 8. Блок-схема алгоритма.

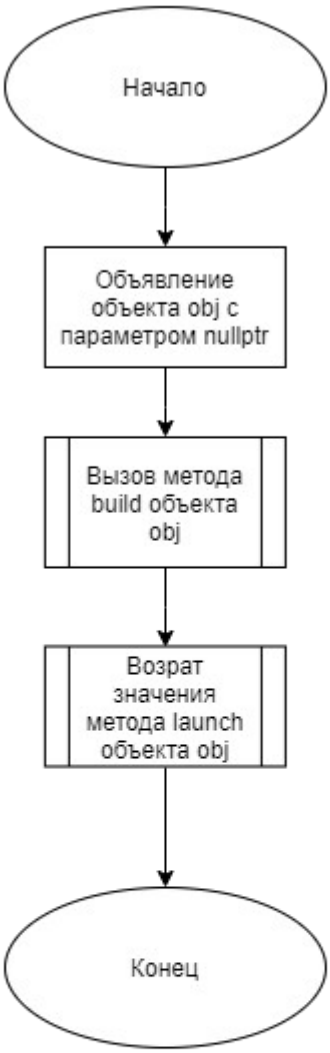

Рис. 9. Блок-схема алгоритма.

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(){ // Изменено
        string parentName, childName;
        int object_number;
        cin >> name;
        while(true){
                cin >> parentName;
                if(parentName == "endtree"){
                        build_ready();
                        break;
                }
                cin >> childName >> object_number;
                if(object_number == 2)
                        Class2* child = new Class2(this-
>getByName(parentName), childName);
                else if(object_number == 3)
                        Class3* child = new Class3(this-
>getByName(parentName), childName);
                else if(object_number == 4)
                        Class4* child = new Class4(this-
>getByName(parentName), childName);
                else if(object_number == 5)
                        Class5* child = new Class5(this-
>getByName(parentName), childName);
                else if(object_number == 6)
                        Class6* child = new Class6(this-
>getByName(parentName), childName);
        }
}
int Application::launch(){
        cout << "Object tree\n";
        print();
        cout << "\nThe tree of objects and their readiness\n";
        print_ready("");
        return 0;
}
```

Файл Application.h

```
#ifndef APPLICATION_H
```

```
#define APPLICATION_H
#include "Tree.h"
#include "Class2.h"
#include "Class3.h"
#include "Class4.h"
#include "Class5.h"
#include "Class6.h"

class Application : public Tree{
public:
        Application(Tree* head);
        void build();
        int launch();
};
#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 Class3.h

```
#ifndef CLASS3_H
#define CLASS3_H
#include "Tree.h"

class Class3 : public Tree{
public:
        Class3(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл Class4.h

```
#ifndef CLASS4_H
#define CLASS4_H
#include "Tree.h"
```

```
class Class4 : public Tree{
public:
        Class4(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл Class5.h

```
#ifndef CLASS5_H
#define CLASS5_H
#include "Tree.h"

class Class5 : public Tree{
public:
        Class5(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл Class6.h

```
#ifndef CLASS6_H
#define CLASS6_H
#include "Tree.h"

class Class6 : public Tree{
public:
        Class6(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;
        this->head = head;
        if(head != nullptr){
                head->children.push_back(this);
        }
        ready = 0;
}

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

```
        }
        Tree* result = nullptr;
        for(int i = 0; i < children.size(); i++){
                result = children[i]->getByName(name);
                if(result != nullptr){
                        return result;
                }
        }
        while(result != nullptr){
                result = head;
                getByName(name);
        }
        return nullptr;
}

// КЛ_3_1
void Tree::set_ready(int input_ready){
        if(input_ready == 0){
                ready = 0;
                for(int i = 0; i < children.size(); i++){
                        children[i]->set_ready(0);
                }
        }
        else{
                Tree* temp = this->head;
                while(temp != nullptr){
                        if(temp->ready == 0){
                                ready = 0;
                                return;
                        }
                        temp = temp->head;
                }
                this->ready = input_ready;
        }
}
void Tree::build_ready(){
        string name;
        int input_ready;
        while(cin >> name >> input_ready){
                if(getByName(name) != nullptr){
                        (getByName(name))->set_ready(input_ready);
                }
        }
}
void Tree::print_ready(string tab) {
        if(ready > 0 || ready < 0)
                cout << get_name() << " is ready";
        else
                cout << get_name() << " is not ready";
        if(children.size() != 0){
                tab += "    ";
                for(Tree* element : children){
                        cout << endl << tab;
                        element->print_ready(tab);
                }
        }
}
```

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;
        int ready; // Готовность объекта
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(int count = 1);

        // КЛ_3_1
        Tree* getByName(string name);
        void set_ready(int input_ready); // Задаёт готовность объекту
        void build_ready(); // Установка готовности объектам
        void print_ready(string tab); // Вывод готовности объектов
};
#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
app_root app_root object_01 3 app_root object_02 2 object_02 object_04 3 object_02 object_05 5 object_01 object_07 2 endtree app_root 1 object_07 3 object_01 1 object_02 -2 object_04 1	Object tree app_root object_01 object_07 object_02 object_04 object_05 The tree of objects and their readiness app_root is ready object_01 is ready object_07 is not ready object_02 is ready object_04 is ready object_05 is not ready	Object tree app_root object_01 object_07 object_02 object_04 object_05 The tree of objects and their readiness app_root is ready object_01 is ready object_07 is not ready object_02 is ready object_04 is ready object_05 is not ready
root root obj1 2 root obj2 3 obj1 obj3 4 endtree root 0 obj1 2 obj2 4 obj3 0	Object tree root obj1 obj3 obj2 The tree of objects and their readiness root is not ready obj1 is not ready obj3 is not ready obj2 is not ready	Object tree root obj1 obj3 obj2 The tree of objects and their readiness root is not ready obj1 is not ready obj3 is not ready obj2 is not ready
root root obj1 2 root obj2 5 root obj3 4 obj3 obj4 3 obj4 obj5 4 obj3 obj6 2 endtree root 2 obj1 12 obj2 4 obj3 -3 obj4 0	Object tree root obj1 obj2 obj3 obj4 obj5 obj6 The tree of objects and their readiness root is ready obj1 is ready obj2 is ready obj3 is ready obj4 is not ready obj5 is not ready obj6 is not ready	Object tree root obj1 obj2 obj3 obj4 obj5 obj6 The tree of objects and their readiness root is ready obj1 is ready obj2 is ready obj3 is ready obj4 is not ready obj5 is not ready obj6 is not ready

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).