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-13-21	Черномуров С.А.
Руководитель практики	Ассистент	Асадова Ю.С.
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

Для решения задачи используются:

- Объекты стандартных потоков ввода и вывода cin и cout соответственно. Используются для ввода с клавиатуры и вывода на экран.
- Оператор цикла со счетчиком for. Используется для вывода дерева иерархии объектов на экран и поиска объектов по их имени.
- Оператор цикла с предусловием while. Используется для постройки и вывода дерева иерархии объектов на экран.
- Условный оператор if .. else. Используется для ветвления алгоритма.
- **Класс Base:**
    - Свойства/поля:
        - Свойства name, parent, children взяты из предыдущей версии приложения.
        - Свойство:
            - Наименование - status;
            - Тип - целочисленный;
            - Модификатор доступа - защищенный.
    - Методы:
        - Методы Base, SetName, SetParent, GetHead, GetName взяты из предыдущей версии приложения.
        - Метод PrintTree доработан на основе предыдущей версии.
        - Метод ObjectByName:
            - Функционал - параметризированный метод, возвращающий указатель на объект класса Base по его имени.
        - Метод PrintTreeAndStatus:

            - Функционал - метод вывода дерева иерархии объектов и отметок их готовности.
        - Метод SetStatus:
            - Функционал - параметризированный метод, устанавливающий готовность объекта.
- **Класс Application:**
    - Методы:
        - Методы BuildTree, StartApp доработаны на основе предыдущей версии.
- **Класс Child2:**
    - Свойства/поля:
        - Унаследованы из класса Base.
    - Методы:
        - Унаследованы из класса Base.
- **Класс Child3:**
    - Свойства/поля:
        - Унаследованы из класса Base.
    - Методы:
        - Унаследованы из класса Base.
- **Класс Child4:**
    - Свойства/поля:
        - Унаследованы из класса Base.
    - Методы:
        - Унаследованы из класса Base.
- **Класс Child5:**
    - Свойства/поля:
        - Унаследованы из класса Base.

    - Методы:
        - Унаследованы из класса Base.
- **Класс Child6:**
    - Свойства/поля:
        - Унаследованы из класса Base.
    - Методы:
        - Унаследованы из класса Base.

Иерархия наследования классов:

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	Base			Базовый класс в иерархии классов. Содержит основные поля и методы		
		Application	public		2	
		Child2	public		3	
		Child3	public		4	
		Child4	public		5	
		Child5	public		6	
		Child6	public		7	
2	Application			Класс корневого объекта (приложения)		
3	Child2			Класс объектов, подчиненных корневому		

				объекту класса Application		
4	Child3			Класс объектов, подчиненных корневому объекту класса Application		
5	Child4			Класс объектов, подчиненных корневому объекту класса Application		
6	Child5			Класс объектов, подчиненных корневому объекту класса Application		
7	Child6			Класс объектов, подчиненных корневому объекту класса Application		

Описание алгоритма

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

Класс объекта: Base

Модификатор доступа: public

Метод: PrintTree

Функционал: Метод вывода дерева иерархии объектов

Параметры: Отсутствуют

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода PrintTree класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление целочисленной переменной с инициализацией i=0	2	Использование переменой i в качестве счетчика
2	Значение i меньше значения, возвращенного методом size объекта children	Объявление целочисленной переменной с инициализацией flag=0	3	
			∅	Выход из цикла
3		Объявление указателя на объект класса Base с	4	

		инициализацией ob=указатель на текущий объект		
4		Объявление строковой переменной s=" "	5	
5	Значение, возвращенное методом GetHead объекта ob не равно nullptr	Инкрементирование flag	6	Вложенный цикл с предусловием
			7	Выход из вложенного цикла с предусловием
6		Присвоение ob значения, возвращенного методом GetHead объекта ob	5	
7		Объявление целочисленной переменной с инициализацией j=0	8	Использование переменой j в качестве счетчика
8	Значение j меньше значения flag	Присвоение значению s значения s+" "	9	
			10	Выход из вложенного цикла со счетчиком
9		Инкрементирование j	8	
10		Вывод на экран переноса на новую строку, s	11	
11		Вывод на экран значения, возвращенного методом GetName объекта children[i]	12	
12		Вызов метода PrintTree объекта children[i]	13	
13		Инкрементирование i	2	

Класс объекта: Application

Модификатор доступа: public

Метод: BuildTree

Функционал: Метод построения исходного дерева иерархии объектов

Параметры: Отсутствуют

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода BuildTree класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление строковой переменной name	2	
2		Считывание с клавиатуры значения name	3	
3		Вызов метода SetName текущего объекта со строковым параметром name	4	
4		Объявление переменной типа данных: указатель на объект класса Base; с инициализацией ob_base=указатель на текущий объект	5	
5		Объявление указателя на динамический объект класса Child2 с инициализацией child2=nullptr	6	
6		Объявление указателя на динамический объект класса Child3 с инициализацией child3=nullptr	7	
7		Объявление указателя на динамический объект класса Child4 с инициализацией	8	

		child4=nullptr		
8		Объявление указателя на динамический объект класса Child5 с инициализацией child5=nullptr	9	
9		Объявление указателя на динамический объект класса Child6 с инициализацией child6=nullptr	10	
10		Создание объекта ukaz класса vector с значениями указателей на объекты класса Base	11	
11		Вызов метода push_back объекта ukaz с параметром ob_base	12	
12		Объявление строковой переменной parentname	13	
13		Объявление строковой переменной childname	14	
14		Объявление целочисленной переменной child_number	15	
15		Считывание с клавиатуры значения parentname	16	
16	Значение parentname не равно "endtree"	Считывание с клавиатуры значений childname, child_number	17	
			17	
17	Значение parentname не равно "endtree"	Присвоение переменной ob_base значения, возвращенного методом ObjectByName с параметрами parentname, ukaz	18	
			30	Выход из цикла
18	Значение child_number равно 2	Создание динамического объекта child2 класса Child2 путем вызова	19	

		параметризированного конструктора с параметрами ob_base, childname		
			20	
19		Вызов метода push_back объекта ukaz с параметром child2	20	
20	Значение child_number равно 3	Создание динамического объекта child3 класса Child3 путем вызова параметризированного конструктора с параметрами ob_base, childname	21	
			22	
21		Вызов метода push_back объекта ukaz с параметром child3	22	
22	Значение child_number равно 4	Создание динамического объекта child4 класса Child4 путем вызова параметризированного конструктора с параметрами ob_base, childname	23	
			24	
23		Вызов метода push_back объекта ukaz с параметром child4	24	
24	Значение child_number равно 5	Создание динамического объекта child5 класса Child5 путем вызова параметризированного конструктора с параметрами ob_base, childname	25	
			26	
25		Вызов метода push_back объекта ukaz с параметром child5	26	
26	Значение child_number равно 6	Создание динамического объекта child6 класса Child6 путем вызова	27	

		параметризированного конструктора с параметрами ob_base, childname		
			28	
27		Вызов метода push_back объекта ukaz с параметром child6	28	
28		Считывание с клавиатуры значения parentname	29	
29	Значение parentname не равно "endtree"	Считывание с клавиатуры значений childname, child_number	17	
			17	
30		Объявление целочисленной переменной status	31	
31	Значения parentname, status считаны с клавиатуры	Присвоение переменной ob_base значения, возвращенного методом ObjectByName с параметрами parentname, ukaz	32	
			33	
32		Вызов метода SetStatus объекта ob_base с параметром status	31	
33		Вывод на экран "Object tree" с последующим переносом на новую строку	34	
34		Вывод на экран значения, возвращенного методом GetName текущего объекта	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: StartApp

Функционал: Метод запуска приложения

Параметры: Отсутствуют

Возвращаемое значение: Целочисленный тип данных - код возврата

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода StartApp класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода PrintTree текущего объекта	2	
2		Вывод на экран переноса на новую строку, "The tree of objects and their readiness", перенос на новую строку, значения, возвращенного методом GetName текущего объекта, " is "	3	
3	Значение свойства status текущего объекта не равно нулю	Вывод на экран "ready"	4	
		Вывод на экран "not ready"	4	
4		Вызов метода PrintTreeAndStatus текущего объекта	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: SetStatus

Функционал: Параметризированный метод, устанавливающий готовность объекта

Параметры: Целочисленный параметр status

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода SetStatus класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	Значение status равно нулю	Объявление целочисленной переменной с инициализацией i=0	2	Использование переменной i в качестве счетчика
		Объявление переменной типа данных: указатель на объект класса Base; с инициализацией ob=указатель на текущий объект	4	
2	Значение i меньше значения, возвращенного методом size объекта children	Вызов метода SetStatus с нулевым параметром объекта children[i]	3	
			6	Выход из цикла
3		Инкрементирование i	2	
4	Значение, возвращенное методом GetHead объекта ob не равно nullptr	Присвоение ob значения, возвращенного методом GetHead объекта ob	5	Цикл с предусловием
			6	Выход из цикла
5	Значение свойства status объекта ob		∅	

	равно нулю			
			4	
6		Присвоение значения status свойству status текущего объекта	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: PrintTreeAndStatus

Функционал: Метод вывода дерева иерархии объектов

Параметры: Отсутствуют

Возвращаемое значение: Отсутствует

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода PrintTreeAndStatus класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление целочисленной переменной с инициализацией i=0	2	Использование переменой i в качестве счетчика
2	Значение i меньше значения, возвращенного методом size объекта children	Объявление целочисленной переменной с инициализацией flag=0	3	
			∅	Выход из цикла
3		Объявление указателя на объект класса Base с инициализацией ob=указатель на текущий	4	

		объект		
4		Объявление строковой переменной s=" "	5	
5	Значения, возвращенное методом GetHead объекта ob не равно nullptr	Инкрементирование flag	6	Вложенный цикл с предусловием
			7	Выход из вложенного цикла с предусловием
6		Присвоение ob значения, возвращенного методом GetHead объекта ob	5	
7		Объявление целочисленной переменной с инициализацией j=0	8	Использование переменой j в качестве счетчика
8	Значение j меньше значения flag	Присвоение значению s значения s+" "	9	
			10	Выход из вложенного цикла со счетчиком
9		Инкрементирование j	8	
10		Вывод на экран переноса на новую строку, s	11	
11		Вывод на экран значения, возвращенного методом GetName объекта children[i], " is "	12	
12	Значение свойства status объекта children[i] не равно нулю	Вывод на экран "ready"	13	
		Вывод на экран "not	13	

		ready"		
13		Вызов метода PrintTreeAndStatus объекта children[i]	14	
14		Инкрементирование i	2	

Функция: main

Функционал: Основной алгоритм программы

Параметры: Отсутствуют

Возвращаемое значение: Целочисленный тип данных - код возврата

Алгоритм функции представлен в таблице 7.

Таблица 7. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Создание объекта app класса Application путем вызова параметризированного конструктора с параметром указателя на объект класса Base, равным nullptr; пустым строковым параметром	2	
2		Вызов метода BuildTree объекта app	3	
3		Возврат функцией значения, возвращенного методом StartApp объекта app	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

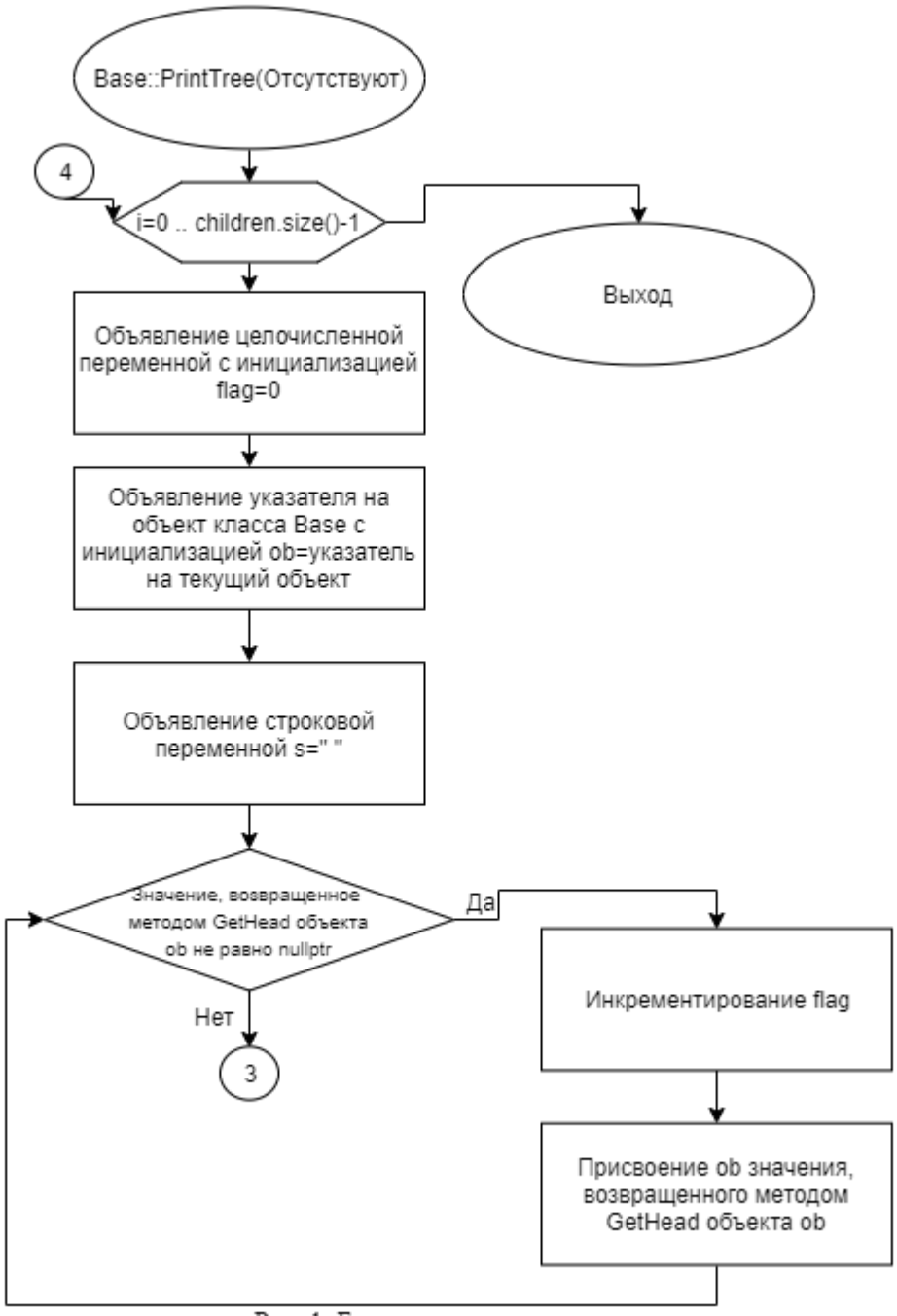

Рис. 1. Блок-схема алгоритма.

Рис. 2. Блок-схема алгоритма.

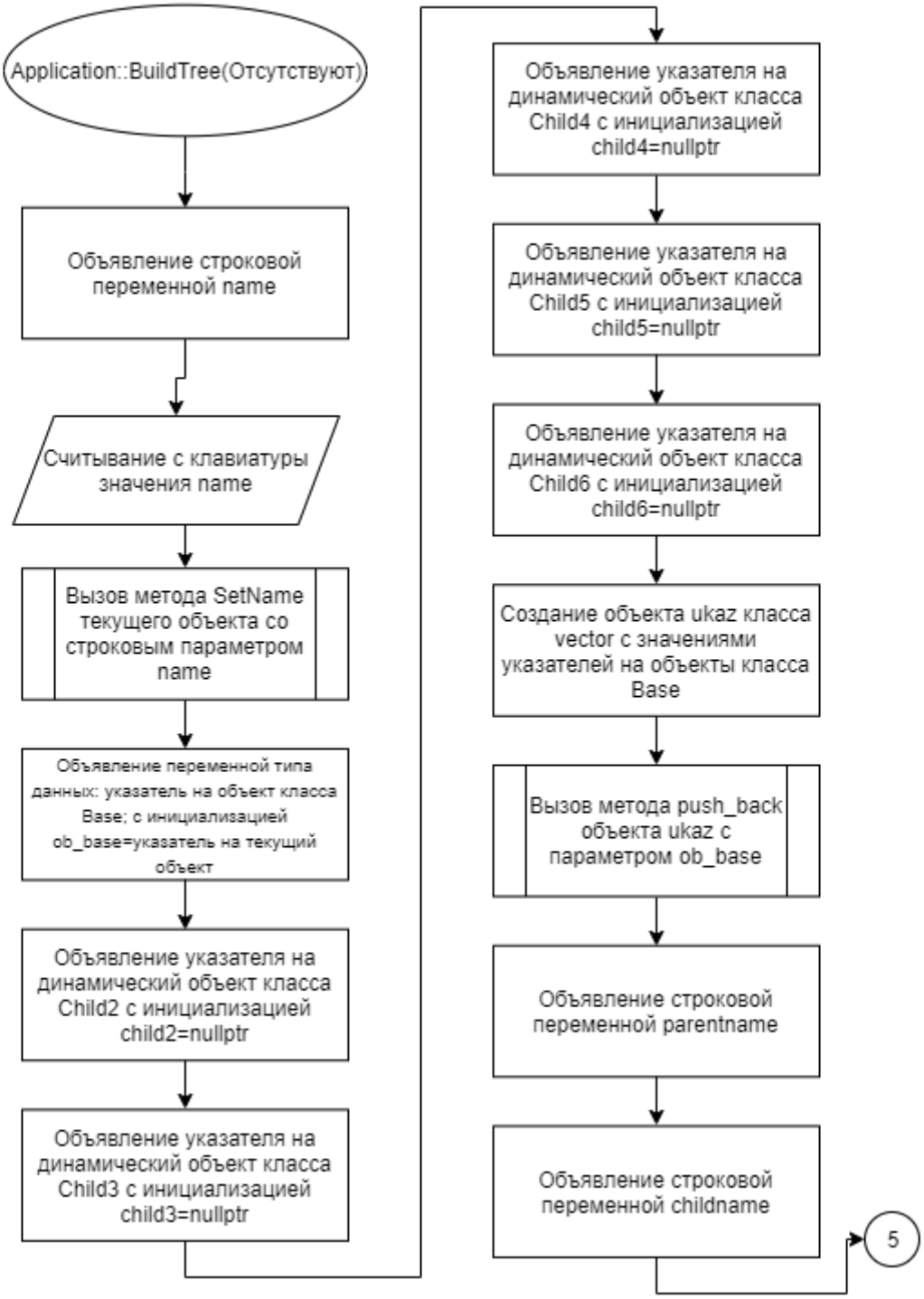

Рис. 3. Блок-схема алгоритма.

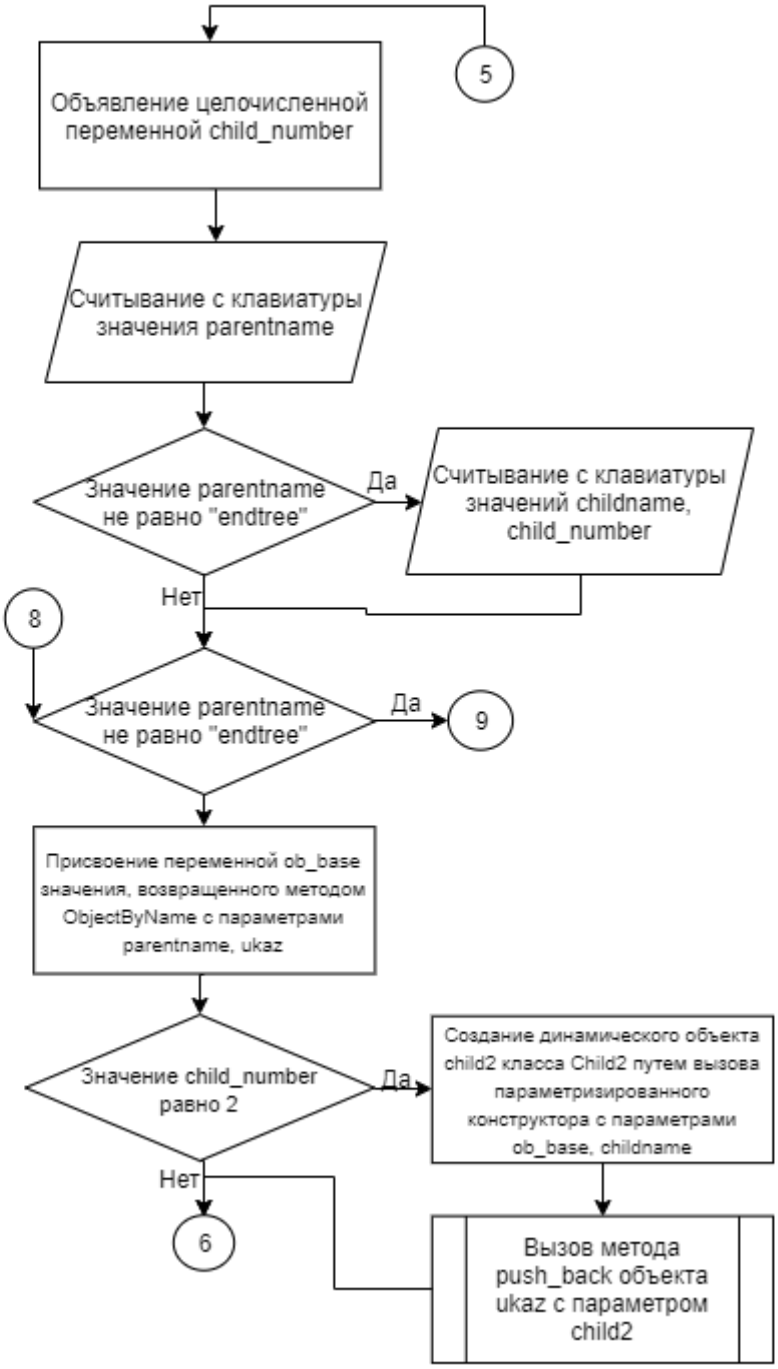

Рис. 4. Блок-схема алгоритма.

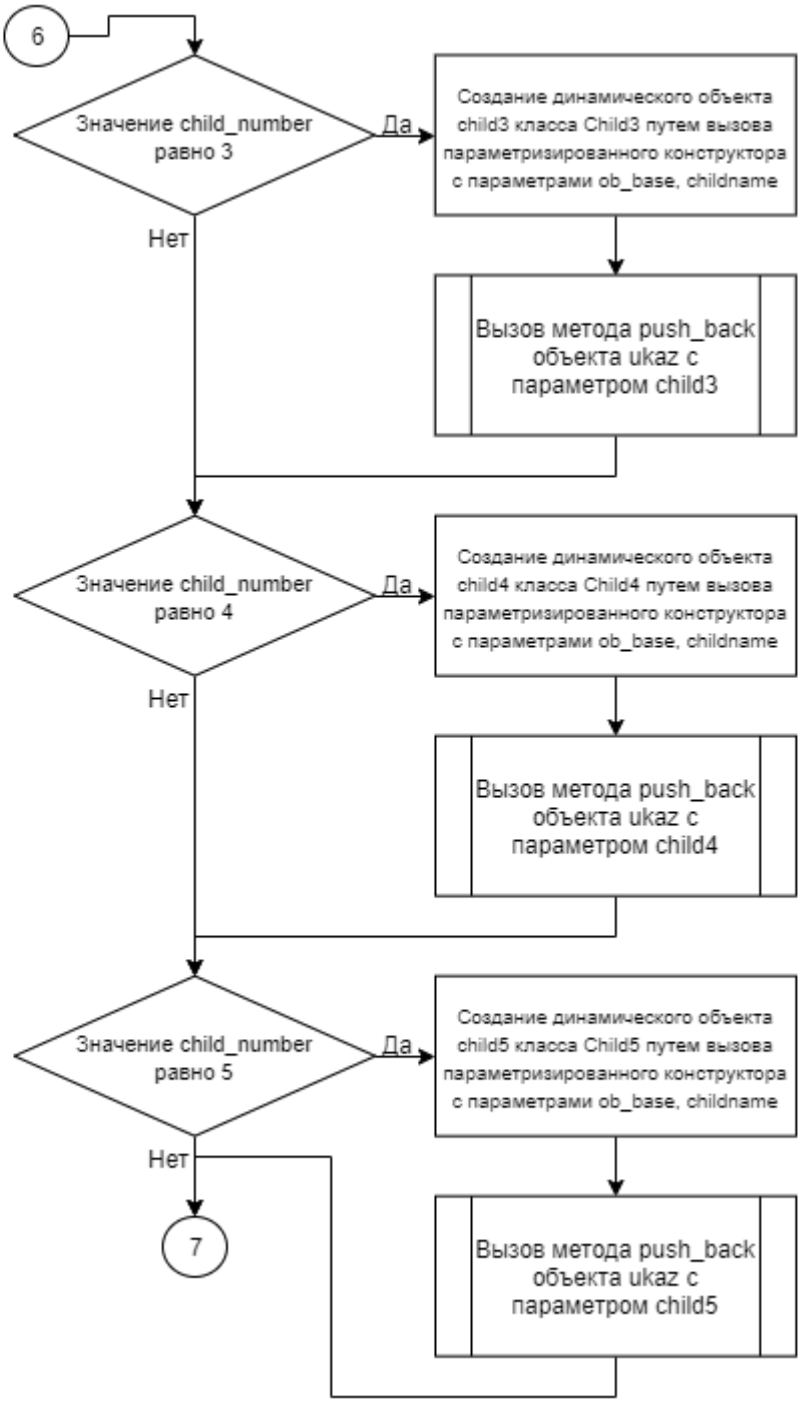

Рис. 5. Блок-схема алгоритма.

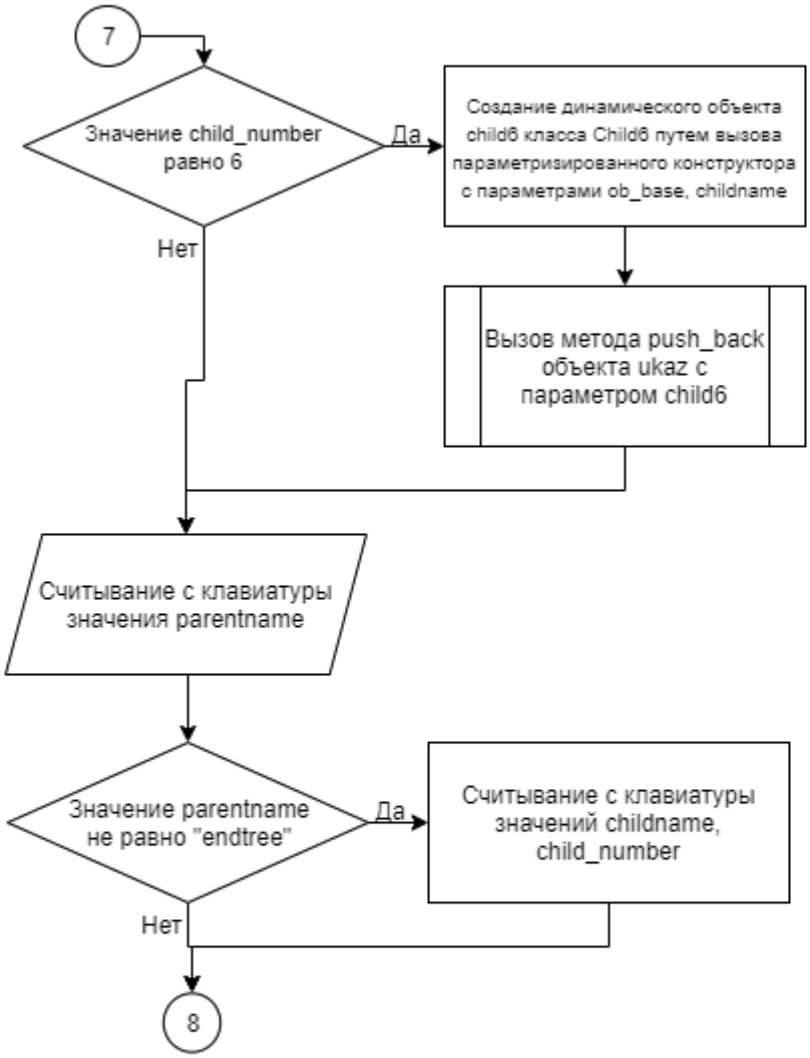

Рис. 6. Блок-схема алгоритма.

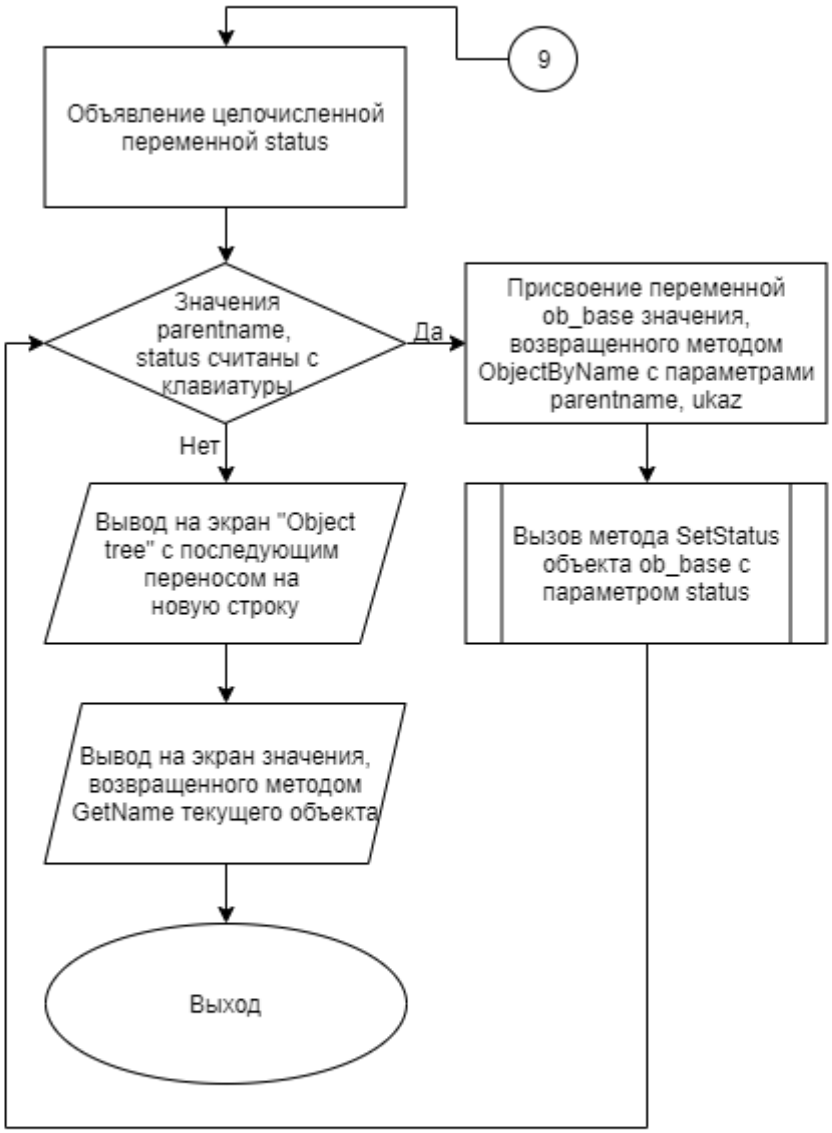

Рис. 7. Блок-схема алгоритма.

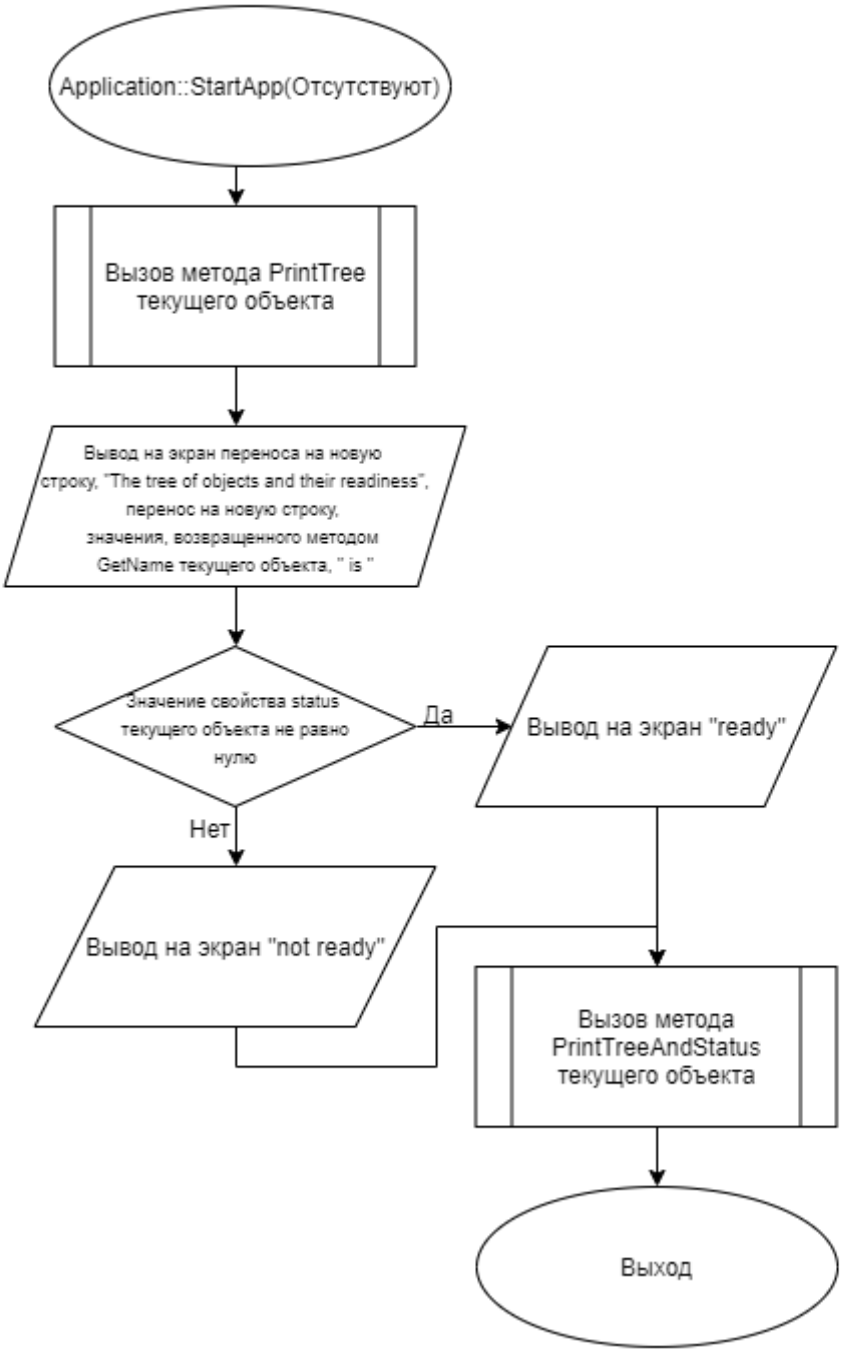

Рис. 8. Блок-схема алгоритма.

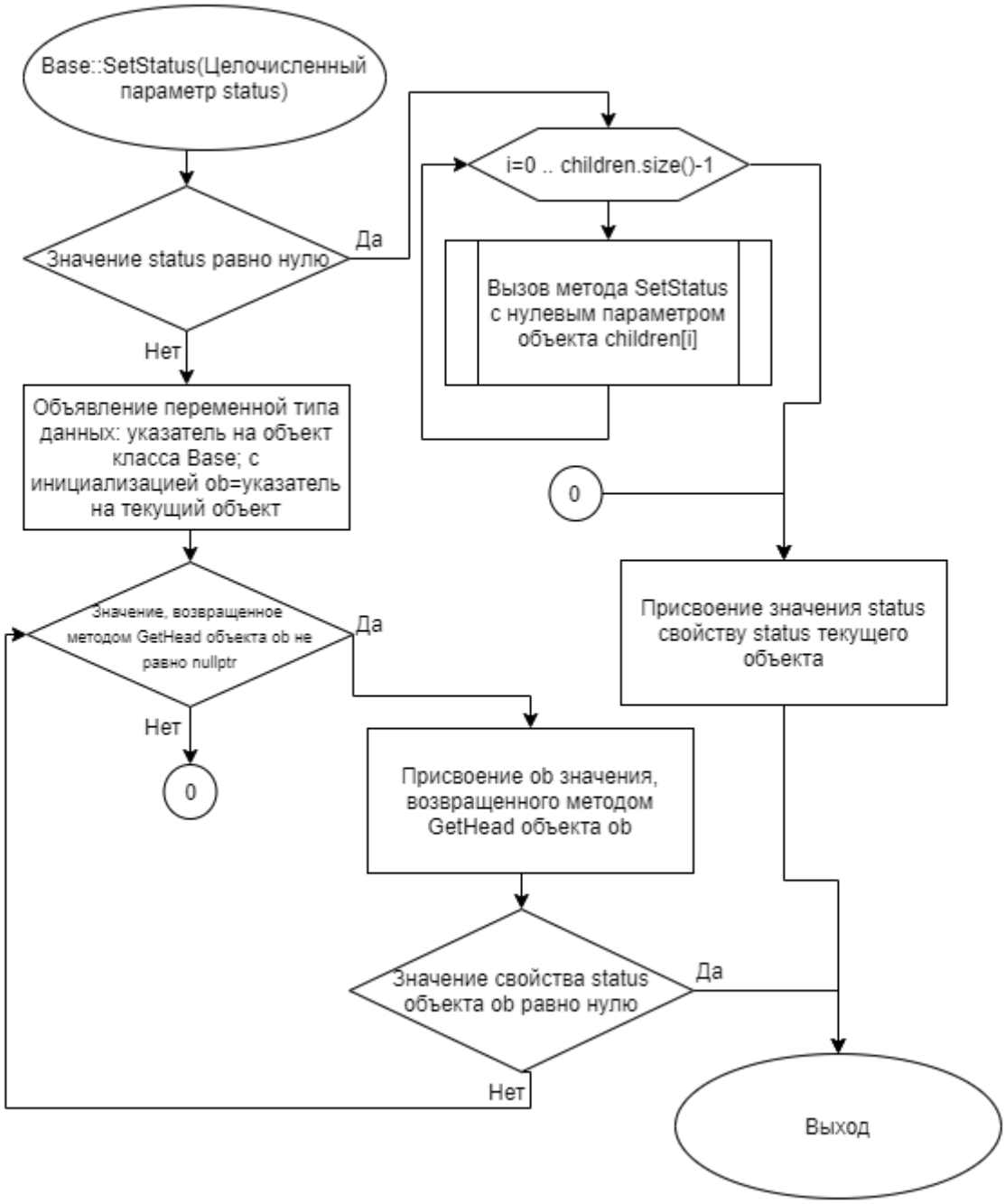

Рис. 9. Блок-схема алгоритма.

Рис. 10. Блок-схема алгоритма.

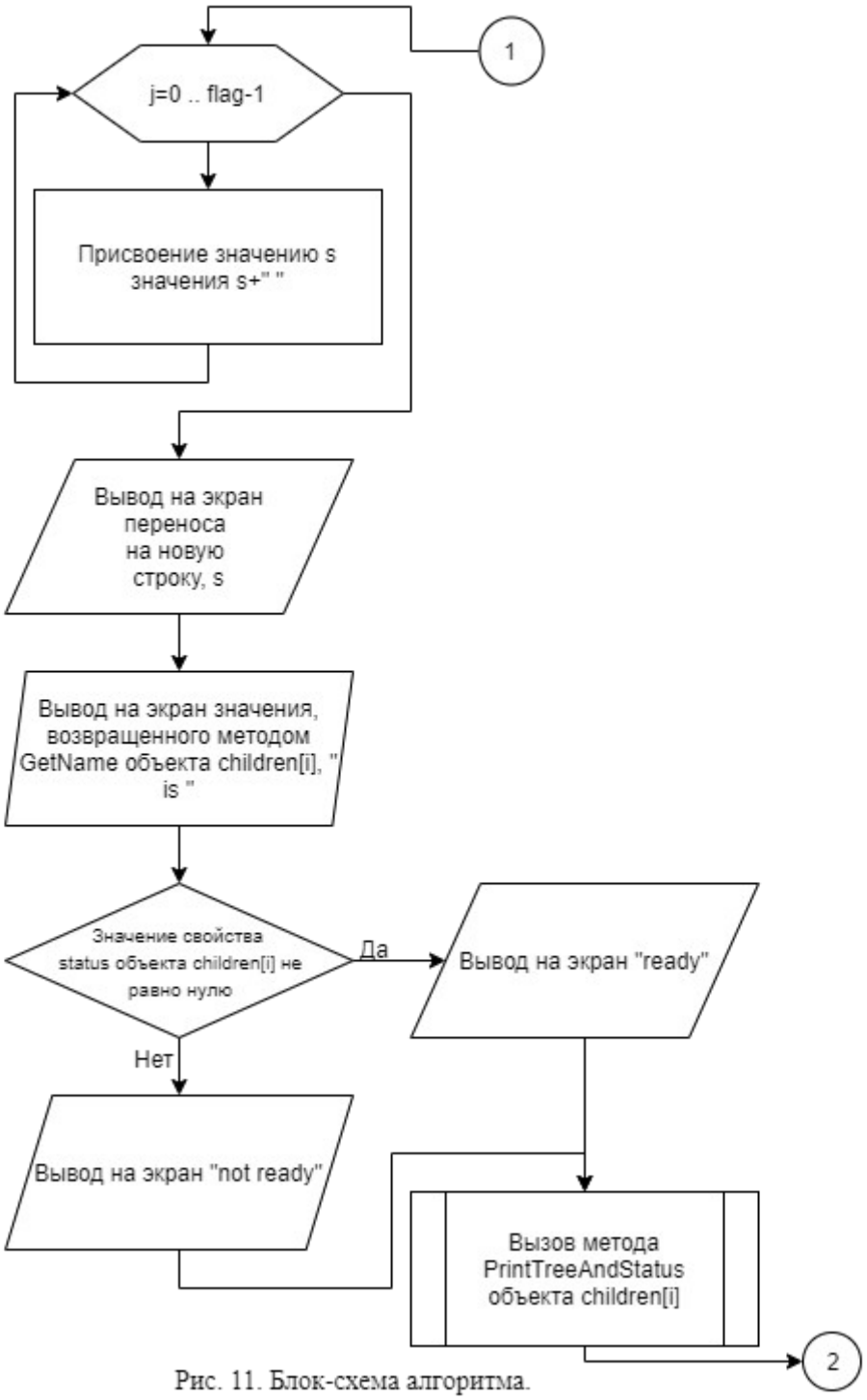

Рис. 11. Блок-схема алгоритма.

Рис. 12. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Application.cpp

```
#include "Application.h"
#include "Child2.h"
#include "Child3.h"
#include "Child4.h"
#include "Child5.h"
#include "Child6.h"
#include <iostream>
using namespace std;

void Application::BuildTree() {
        string name;
        cin >> name;

        this->SetName(name);

        Base* ob_base=this;
        Child2* child2=nullptr;
        Child3* child3=nullptr;
        Child4* child4=nullptr;
        Child5* child5=nullptr;
        Child6* child6=nullptr;


        vector <Base*> ukaz;
        ukaz.push_back(ob_base);

        string parentname;
        string childname;
        int child_number;

        cin>>parentname;
                if (parentname!="endtree")
                        cin>>childname>>child_number;

        while (parentname != "endtree") {
                ob_base = ObjectByName(parentname, ukaz);

                if (child_number == 2) {child2=new Child2(ob_base,childname);
ukaz.push_back(child2);}
                else
                if (child_number == 3) {child3=new Child3(ob_base,childname);
ukaz.push_back(child3);}
                else
                if (child_number == 4) {child4=new Child4(ob_base,childname);
ukaz.push_back(child4);}
```

```
                else
                if (child_number == 5) {child5=new Child5(ob_base,childname);
ukaz.push_back(child5);}
                else
                if (child_number == 6) {child6=new Child6(ob_base,childname);
ukaz.push_back(child6);}

                cin>>parentname;
                if (parentname!="endtree")
                        cin>>childname>>child_number;

        }

        int status;
        while (cin >> parentname >> status){
                ob_base = ObjectByName(parentname, ukaz);
                ob_base->SetStatus(status);
        }

        cout<<"Object tree\n";
        cout << this->GetName();// << this->GetName() << "  ";
}

int Application::StartApp() {
        this->PrintTree();
        cout<<"\nThe tree of objects and their readiness\n"<<this-
>GetName()<<" is ";
        if (this->status!=0) cout<<"ready";
        else cout<<"not ready";
        this->PrintTreeAndStatus();

        return 0;
}
```

Файл Application.h

```
#ifndef APP_H
#define APP_H

#include "Base.h"

class Application : public Base {
        using Base::Base;
public:
        void BuildTree();// доработан
        int StartApp();// доработан
};

#endif
```

Файл Base.cpp

```
#include "Base.h"
#include <iostream>
using namespace std;


Base::Base(Base* parent, string name) {
	SetName(name);
	SetParent(parent);
	if (parent) { // !=nullptr
		parent->children.push_back(this);
	}
}

void Base::SetName(string name) {
	this->name = name; // наименование объекта
}

string Base::GetName() {
	return name; // возврат имени объекта
}

void Base::PrintTree() {


	for (int i = 0; i < children.size(); i++) {
		int flag=0;
		Base* ob=this;
		string s="    ";

		while (ob->GetHead()!=nullptr){
			flag++;
			ob=ob->GetHead();
		}

		for (int j=0;j<flag;j++)
			s+="    ";

		cout<<"\n"<<s;

		cout<<children[i]->GetName();

		children[i]->PrintTree();
	}
	}


Base* Base::GetHead() {
	return  parent; // возврат указателя на головной объект
}

void Base::SetParent(Base* parent) {
	this->parent = parent;
}

Base* Base::ObjectByName(string name, vector <Base*> &vec){ //получение
объекта по имении в дереве иерархи
	for (int i=0;i<vec.size();i++){
```

```
                if (vec[i]->GetName()==name) return vec[i];
        }
        return nullptr;
}

void Base::SetStatus(int status){
        if (status==0){
                for (int i = 0; i < children.size(); i++)
                        children[i]->SetStatus(0);
        }
        else{
                Base* ob=this;
                while (ob->GetHead()!=nullptr){
                        ob=ob->GetHead();
                        if (ob->status==0) return;
                }
                this->status=status;
        }
}

void Base::PrintTreeAndStatus(){
        for (int i = 0; i < children.size(); i++) {
                int flag=0;
                Base* ob=this;
                string s="    ";

                while (ob->GetHead()!=nullptr){
                        flag++;
                        ob=ob->GetHead();
                }

                for (int j=0;j<flag;j++)
                        s+="    ";

                cout<<"\n"<<s;

                cout<<children[i]->GetName()<<" is ";
                if (children[i]->status!=0) cout<<"ready";
                else cout<<"not ready";

                children[i]->PrintTreeAndStatus();
        }
}
```

Файл Base.h

```
#ifndef BASE_H
#define BASE_H

#include <string>
#include <vector>
using namespace std;

class Base {
        protected:
                string name;
```

```
                int status=0; // новое
        private:
                Base* parent;
                vector <Base*> children;
        public:
                Base(Base* parent, string name);

                void SetName(string name);
                void PrintTree(); // доработано
                void SetParent(Base* parent);

                Base* GetHead();
                Base* ObjectByName(string name, vector <Base*> &vec);

                string GetName();

                void PrintTreeAndStatus(); // новое
                void SetStatus(int status); // новое
};

#endif
```

Файл Child2.h

```
#ifndef CHILD2_H
#define CHILD2_H

#include "Application.h"

class Child2 : public Base {
        using Base::Base;
};

#endif
```

Файл Child3.h

```
#ifndef CHILD3_H
#define CHILD3_H

#include "Application.h"

class Child3 : public Base {
        using Base::Base;
};

#endif
```

Файл Child4.h

```
#ifndef CHILD4_H
#define CHILD4_H

#include "Application.h"

class Child4 : public Base {
        using Base::Base;
};

#endif
```

Файл Child5.h

```
#ifndef CHILD5_H
#define CHILD5_H

#include "Application.h"

class Child5 : public Base {
        using Base::Base;
};

#endif
```

Файл Child6.h

```
#ifndef CHILD6_H
#define CHILD6_H

#include "Application.h"

class Child6 : public Base {
        using Base::Base;
};

#endif
```

Файл main.cpp

```
#include "Base.h"
#include "Application.h"
#include "Child2.h"
```

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
        Application app(nullptr, "");
        app.BuildTree();
        return app.StartApp();
}
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root root ob1 3 root ob2 4 root ob3 5 ob1 ob4 6 ob2 ob10 2 ob2 ob11 2 ob3 ob6 4 ob6 ob7 5 ob7 ob9 6 ob7 ob8 5 endtree ob1 1 ob2 1 ob3 1 ob6 0 ob7 1 ob8 0 ob9 1 ob10 1	Object tree root ob1 ob4 ob2 ob10 ob11 ob3 ob6 ob7 ob9 ob8 The tree of objects and their readiness root is not ready ob1 is not ready ob4 is not ready ob2 is not ready ob10 is not ready ob11 is not ready ob3 is not ready ob6 is not ready ob7 is not ready ob9 is not ready ob8 is not ready	Object tree root ob1 ob4 ob2 ob10 ob11 ob3 ob6 ob7 ob9 ob8 The tree of objects and their readiness root is not ready ob1 is not ready ob4 is not ready ob2 is not ready ob10 is not ready ob11 is not ready ob3 is not ready ob6 is not ready ob7 is not ready ob9 is not ready ob8 is not ready
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 ob1 3 root ob2 4 root ob3 5 ob1 ob4 6 ob2 ob10 2 ob2 ob11 2 ob3 ob6 4 ob6 ob7 5 ob7 ob9 6 ob7 ob8 5 endtree root 1 ob1 1 ob2 1 ob3 1 ob6 0	Object tree root ob1 ob4 ob2 ob10 ob11 ob3 ob6 ob7 ob9 ob8 The tree of objects and their readiness root is ready ob1 is ready ob4 is not ready ob2 is	Object tree root ob1 ob4 ob2 ob10 ob11 ob3 ob6 ob7 ob9 ob8 The tree of objects and their readiness root is ready ob1 is ready ob4 is not ready ob2 is

ob7 1 ob8 0 ob9 1 ob10 1	ready ob10 is ready ob11 is not ready ob3 is ready ob6 is not ready ob7 is not ready ob9 is not ready ob8 is not ready	ready ob10 is ready ob11 is not ready ob3 is ready ob6 is not ready ob7 is not ready ob9 is not ready ob8 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).