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 дисциплины)

Тема курсовой работы Разработка программы для поиска объекта на дереве иерархии по координате

Студент группы ИВБО-04-20 Манохин Дмитрий Александрович _____
(учебная группа, фамилия, имя отчество, студента) (подпись студента)

Руководитель курсовой работы Ст. Преподаватель Грач Е.П. _____
(должность, звание, ученая степень) (подпись руководителя

Рецензент (при наличии) _____ _____
(должность, звание, ученая степень) (подпись рецензента)

Работа представлена к защите « ____ » _____ 2021г.

Допущен к защите « ____ » _____ 2021г.

Москва 2021 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой _____
Подпись
Платонова О.В.
ФИО

«25» февраля 2021 г.

ЗАДАНИЕ

на выполнение курсовой работы по дисциплине

«Объектно-ориентированное программирование»

Студент Манохин Дмитрий Александрович Группа ИВБО-04-20
Тема: Разработка программы для поиска объекта на дереве иерархии по координате

Исходные данные: Исходная иерархия расположения объектов, координаты искомых объектов
Перечень вопросов, подлежащих разработке, и обязательного графического материала:
1. Реализовать алгоритм размещения всех объектов в составе программы на иерархическом дереве объектов.
2. Реализовать алгоритм поиска объекта на дереве иерархии.
3. Блок-схема реализованных алгоритмов.

Срок представления к защите курсовой работы: **до «31» мая 2021г.**

Задание на курсовую работу выдал _____ (Грач Е.П.)
Подпись руководителя *Ф.И.О. руководителя*

Задание на курсовую работу получил «25» февраля 2021г
_____ (Манохин Д.А.)
Подпись обучающегося Ф.И.О. исполнителя

Москва 2021 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Ещё в начале 80-х годов прошлого века был изобоетен такой язык программирования как C++, но до сих пор он остается очень востребованным среди многих программистов. И вот почему.

Во-первых, он быстр. Да, у, например, C#, Java и конечно же Python скорость написания кода на порядок выше будет и визуально они занимают намного меньше места, нежели программы написанные на языке программирования C++. Но что же будет важнее для конечного пользователя работы программиста: время разработки приложения или его медленная работа? Ответ конечно же очевиден.

Во-вторых, он универсален. Достаточно просто вспомнить, где используется данный язык программирования: системы моделирования, микроконтроллеры, десктопные и мобильные приложения, игры, роботы, веб - прогнозирования, обработки статистики и в нейронных сетях. Везде. Просто не может существовать такой области в программировании, где C++ был бы бесполезен для использования. Его используют частично почти для любого крупнного проекта, поэтому знание данного языка программирования всегда будет являться только плюсом для программиста, в резюме которого он присутствует.

В-третьих, он активно поддерживается. C++ имеют огромное сообщество программистов, которые регулярно деляться разными библиотеками, шаблонами и их кодами, а также приходят на помощь новичкам в программировании и даже своим опытным коллегам. По C++ есть несколько полезных книг, по которым училось не одно поколение, есть новые, которые учитывают все новые изменения и актуальное ПО, помимо всего

вышеперечисленного есть масса интернет - ресурсов для обучения.

В-четвёртых, он полезен в качестве основы для обучения. JavaScript, Java, C#, как и огромное количество остальных популярных языков программирования, содержат в своей основе принципы и методы C++. Например, принцип работы языка программирования Java, одного из мощнейших языков современности на данный момент, достаточно сложно понять сразу, если не начать своё обучение с его основ, которые впервые появились именно в языке C++. Почти все популярные языки программирования, которые были придуманы позднее, имеют более простую структуру и механизмы нежели C++, а процесс изучения данного языка программирования сводится лишь к освоению его синтаксиса.

В-пятых, он достаточно востребован. Вся вышеописанная информация является причиной того, что программисты, которые изучали язык программирвания C++, востребованы не только в своей стране, но и по всему миру, что не в последнюю очередь сказывается также и на их заработных платах.

Именно по вышеперечисленным причинам язык программирования C++ так актуален сейчас и скорее всего будет ещё актуален не одно десятилетие.

Постановка задачи

Определение указателя на объект по его координате

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

В составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов.
В качестве параметра методу передать путь объекта от корневого. Путь задать в следующем виде:

/root/ob_1/ob_2/ob_3

Уникальность наименования требуется только относительно множества подчиненных объектов для любого головного объекта.

Если система содержит объекты с уникальными именами, то в методе реализовать определение указателя на объект посредством задания координаты в виде:

//«наименование объекта»

Состав и иерархия объектов строится посредством ввода исходных данных.
Ввод организован как в контрольной работе № 1.
Единственное различие: в строке ввода первым указать не наименование головного объекта, а путь к головному объекту.

Подразумевается, что к моменту ввода очередной строки соответствующая ветка на дереве иерархии уже построена.
Система содержит объекты пяти классов, не считая корневого. Номера классов: 2, 3, 4, 5, 6.

Пример ввода иерархии дерева объектов.

root
/root object_1 3 1
/root object_2 2 1
/root/object_2 object_4 3 -1
/root/object_2 object_5 4 1
/root object_3 3 1
/root/object_2 object_3 6 1
/root/object_1 object_7 5 1
/root/object_2/object_4 object_7 3 -1
endtree

Описание входных данных
Множество объектов, их характеристики и расположение на дереве иерархии.

Структура данных для ввода согласно изложенному в фрагменте методического указания [3] в контрольной работе № 1.
После ввода состава дерева иерархии построчно вводятся координаты искомых объектов.
Ввод завершается при вводе: //

Описание выходных данных
Первая строка:

Object tree

Со второй строки вывести иерархию построенного дерева.

Далее, построчно:
«координата объекта» Object name: «наименование объекта»
Разделитель один пробел.

Если объект не найден, то вывести:
«координата объекта» Object not found
Разделитель один пробел.

Метод решения

Используются потоки ввода/вывода cin/cout, условный оператор if, оператор цикла while, оператор цикла for, библиотеки <iostream>, <string> и <vector>.

Иерархия наследования отображена в таблице 1.

Таблица 1. <<Описание иерархии наследования классов>>

№	Имя класса	Классы - наследники	Модификатор доступа	Описание	Номер	Комментарий
1	Base			Базовый класс. Содержит основные поля и методы.		
		Appl	public		2	
		Novel	public		3	
		Cl_2	public		4	
		Cl_3	public		5	
		Cl_4	public		6	
		Cl_5	public		7	
2	Appl			Класс корневого объекта		
3	Novel			Класс объектов подчиненных корневому объекту класса Appl		
4	Cl_2			Класс объектов подчиненных корневому объекту класса Appl		

5	Cl_3			Класс объектов подчиненных корневому объекту класса Appl		
6	Cl_4			Класс объектов подчиненных корневому объекту класса Appl		
7	Cl_5			Класс объектов подчиненных корневому объекту класса Appl		

Класс Base

Функционал:

• objectWay() - string, используется для определения пути до объекта;

• getPar() - Base*, используется для получения указателя на объект по его пути;

• findObj() - string, испоьзуется для нохождения имени объекта по его пути.

Класс Appl

Функционал:

• bild_tree_objects() - void, используется для построения дерева иерархии;

• exec_app() - int, используется для запуска алгоритма приложения.

Описание алгоритма

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

Класс объекта: Appl

Модификатор доступа: public

Метод: bild_tree_objects

Функционал: Построение дерева иерархии объектов

Параметры: -

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода bild_tree_objects класса Appl

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление указателя на объекты класса Base t_child. Инициализация указателей класса Base: t_parent, h_parent, указателем на текущий объект	2	
2		Объявление переменных строкового типа childName и parentName, а также типа int: nclass и kready.	3	
3		Ввод значения parentName	4	

4		Вызов метода setName() для текущего(корневого) объекта с параметром parentName и метода setReady() с параметром 1	5	
5		Ввод значения parentName	6	
6	Имя головного объекта совпадает с "endtree"	Возврат	∅	parentName == "endtree"
		Ввод значений: childName, nclass, kready	7	
7		Присвоение t_parent возврата метода getPar() для корневого объекта с параметром parentName	8	
8	Номер класса принадлежности объекта = 2	Создание нового объекта класса Novel с помощью конструктора с параметрами t_parent и childName, указатель на объект сохраняем в переменную t_child корневого объекта	13	nclass == 2
			9	
9	Номер класса принадлежности объекта = 3	Создание нового объекта класса Cl_2 с помощью конструктора с параметрами t_parent и childName, указатель на объект сохраняем в переменную t_child корневого объекта	13	nclass == 3
			10	
10	Номер класса	Создание нового	13	nclass == 4

	принадлежности объекта = 4	объекта класса Cl_3 с помощью конструктора с параметрами t_parent и childName, указатель на объект сохраняем в переменную t_child корневого объекта		
			11	
11	Номер класса принадлежности объекта = 5	Создание нового объекта класса Cl_4 с помощью конструктора с параметрами t_parent и childName, указатель на объект сохраняем в переменную t_child корневого объекта	13	nclass == 5
			12	
12	Номер класса принадлежности объекта = 6	Создание нового объекта класса Cl_5 с помощью конструктора с параметрами t_parent и childName, указатель на объект сохраняем в переменную t_child корневого объекта	13	nclass == 6
			13	
13		Вызов метода setReady() для только что созданного объекта с параметром kready	5	t_child -> setReady (kready)

Класс объекта: Appl

Модификатор доступа: public

Метод: exec_app

Функционал: Запуск основного алгоритма

Параметры: -

Возвращаемое значение: int, корректность завершения работы

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода exec_app класса Appl

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление переменной строкового типа str	2	
2		Вывод Object tree и имени текущего(корневого) объекта с новой строки	3	
3		Вызов метода printTree() для текущего объекта с параметром 1	4	this -> printTree (1)
4		Ввод значения переменной str	5	
5	str не равно "//"		6	
			8	
6	Возвращаемый указатель методом getPar() с параметром str не нулевой	Вывод с новой строки значения str и " Object name: " и имя объекта, указатель которого получили с помощью метода getPar()	7	this -> getPar(str) this -> getPar(str) -> getName()
		Вывод с новой строки значения str и " Object not found"	7	
7		Ввод значения переменной str	5	
8		Возврат нуля	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: objectWay

Функционал: Возврат пути до объекта

Параметры: Base* t_parent - указатель на объект класса Base

Возвращаемое значение: string, путь до объекта

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода objectWay класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	Указатель на родителя объекта t_parent нулевой	Возврат "/" и имя объекта t_parent	∅	(!t_parent -> parent)
		Возврат вызова метода objectWay() с парамером указателя на родителя объекта t_parent и "/" и имя объекта t_parent	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: getPar

Функционал: Возврат указателя на объект по его пути

Параметры: string parentName - путь до объекта

Возвращаемое значение: Base* - указатель на объект класса Base

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода getPar класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	0-ой и 1-ый элемент parentName равны "/"	Возврат работы метода findParent() с параметром выделенной строки parentName с 2-го индекса до конца	∅	
		Возврат работы метода findParent() с параметром результата работы метода findObj() с параметром parentName	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: findObj

Функционал: Возвращает имя объекта по его пути

Параметры: string parentName - путь до объекта

Возвращаемое значение: string - имя объекта по его пути

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода findObj класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление целочисленных переменных x,y и z, а также указатель на объект класса Base: t_parent	2	
2	Индекс на	Присвоение t_parent	3	

	найденный символ "/" в строке parentName начиная с 1-го индекса меньше длины parentName	работы метода findParent() с параметром выделенной строки parentName с 1-го индекса до первого найденного символа "/" начиная с 1-го индекса		
		Присвоение t_parent работы метода findParent() с параметром выделенной строки parentName с 1-го индекса до конца	3	
3		Присвоение parentName выделенной строки parentName с 1-го индекса до конца	4	
4	Индекс на найденный символ "/" в строке parentName меньше длины parentName	Присвоение переменной r значения -1 Присвоение переменной x значение индекса на найденный символ "/" в строке parentName начиная с 1-го индекса Присвоение переменной y значение индекса на найденный символ "/" в строке parentName начиная с (x + 1) индекса Инициализация переменной i = 0	5	
			9	

5	i меньше размера списка children объекта t_parent		6	
			7	
6	Имя i-го подчиненного объекта, объекта t_parent, совпадает с выделенной строкой parentName, длиной равной: длине строки минус значение переменной y и минус 1 начиная с индекса x	Присвоение переменной r значения 1	5	
			5	
7	Значение перемееной r равно -1	Возврат пусой строки	∅	
		Присвоение t_parent возврата метода findParent() с параметром выделенной строкои parentName, длиной равной: длине строки минус значение переменной y и минус 1 начиная с индекса x	8	
8		Присвоение перменной parentName значения строки parentName с вырезанной частью начиная с 0 до индекса (x + 1)	4	
9		Возврат значения parentName	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

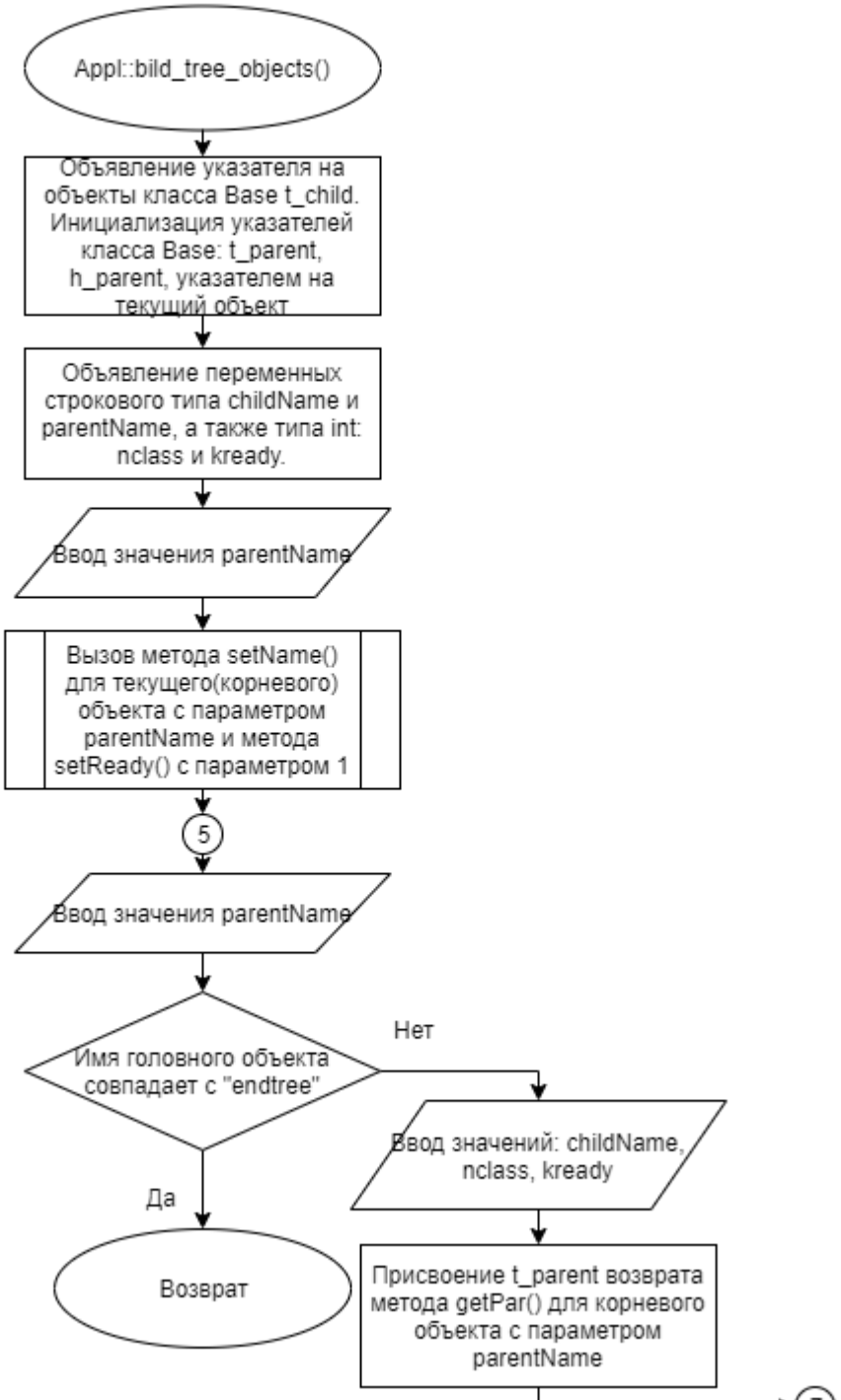

Рис. 1. Блок-схема алгоритма.

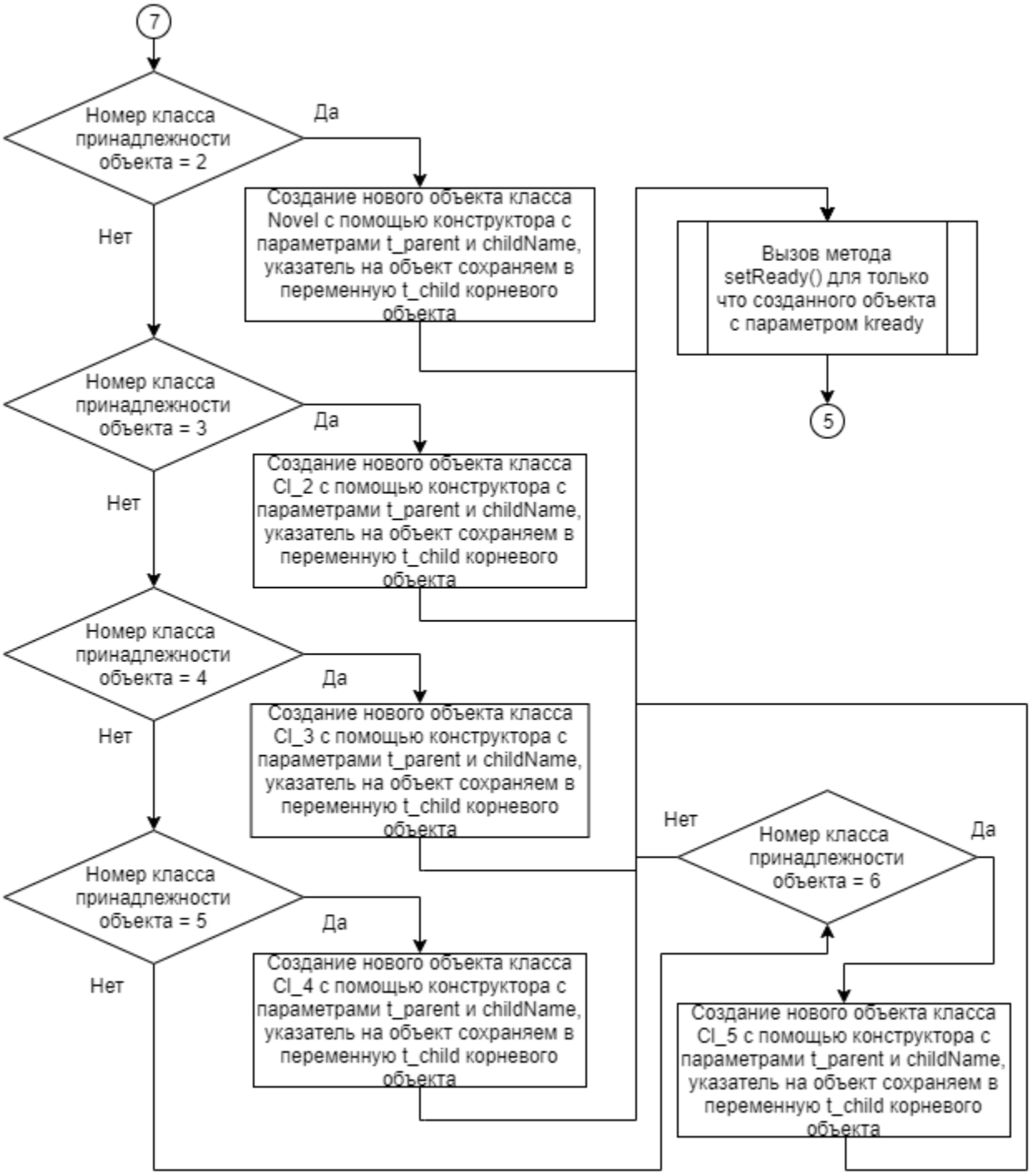

Рис. 2. Блок-схема алгоритма.

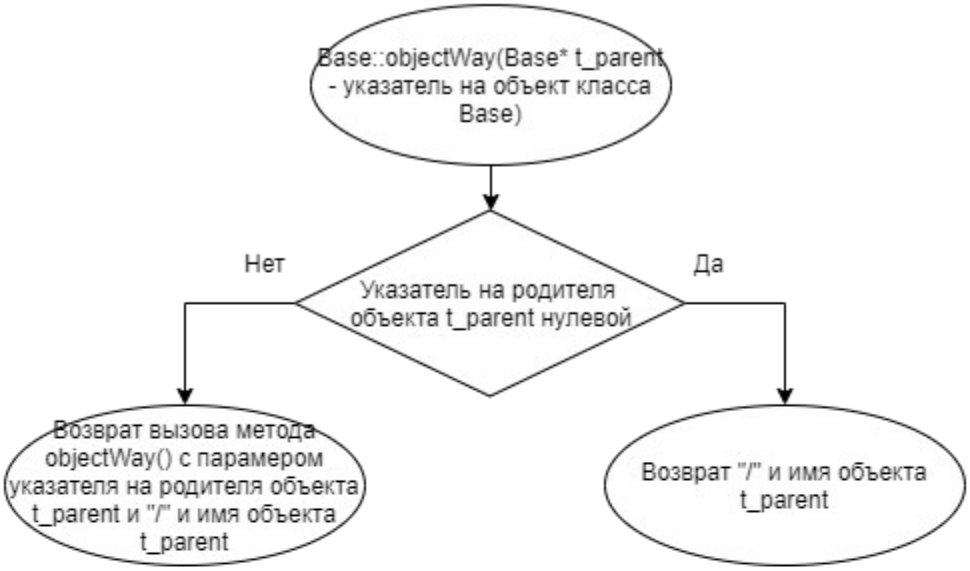

Рис. 3. Блок-схема алгоритма.

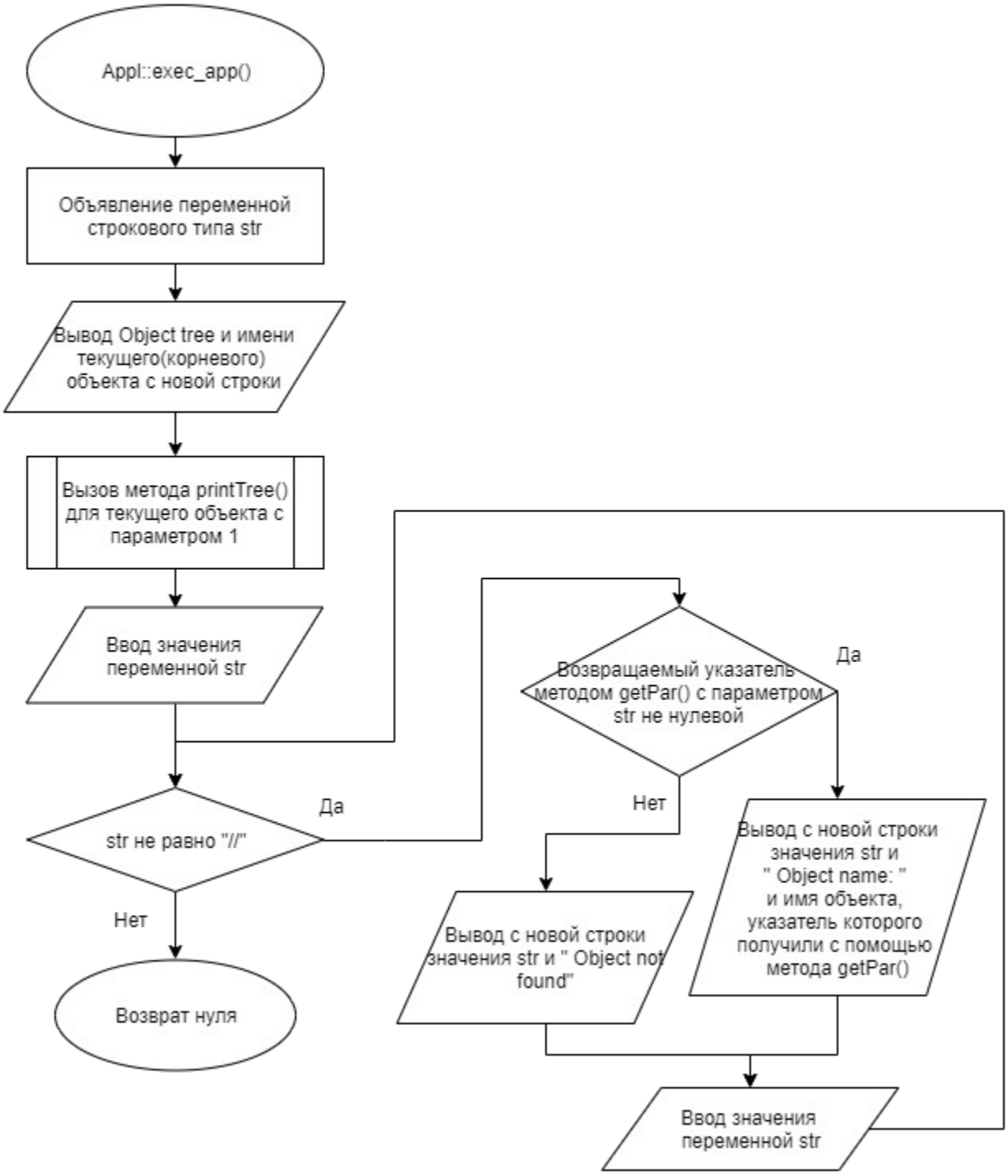

Рис. 4. Блок-схема алгоритма.

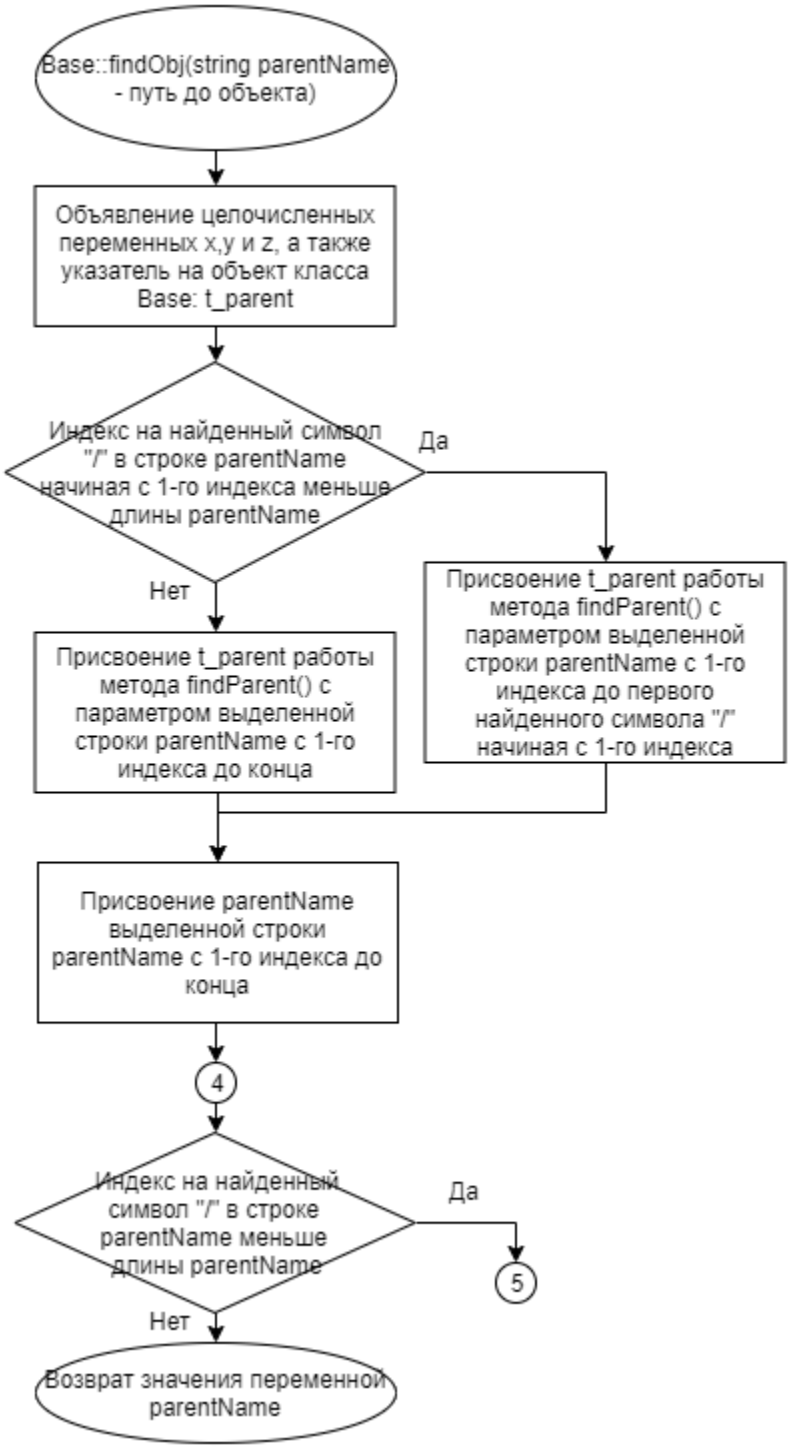

Рис. 5. Блок-схема алгоритма.

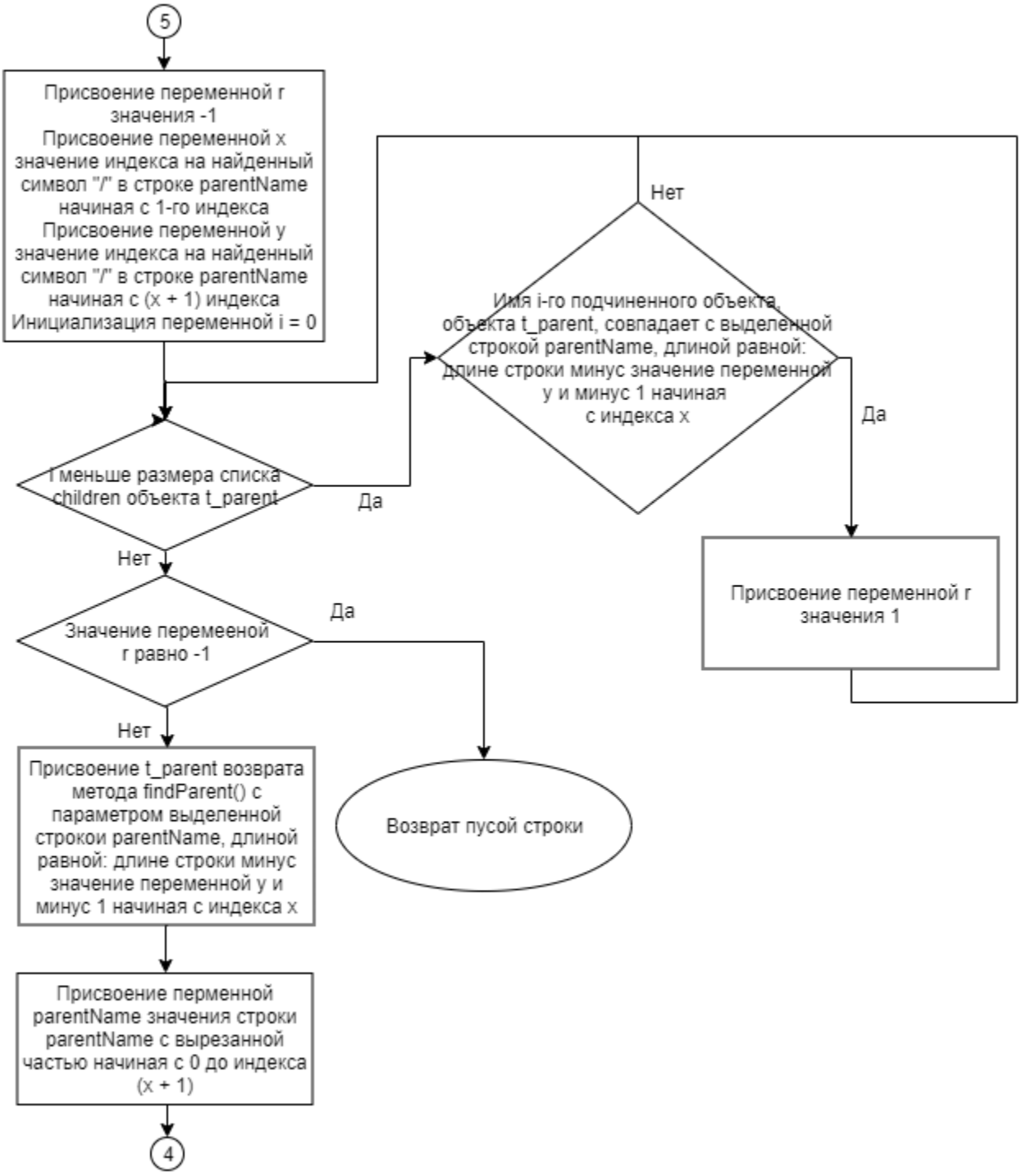

Рис. 6. Блок-схема алгоритма.

Рис. 7. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Appl.cpp

```
#include "Appl.h"
#include "Novel.h"
#include "Cl_2.h"
#include "Cl_3.h"
#include "Cl_4.h"
#include "Cl_5.h"

#include <iostream>
using namespace std;
Appl :: Appl (Base* parent) : Base (parent) {} //Конструктор

void Appl :: bild_tree_objects () //Построение дерева
{
        Base* t_child;
        Base* h_parent = this;
        Base* t_parent = this;
        string parentName, childName;
        int nclass, kready;
        cin >> parentName;
        setName (parentName);
        setReady(1);
        do
        {
                cin >> parentName;
                if ( (parentName == "endtree") /*|| (parentName ==
childName)*/ )
                        return;
                cin >> childName >> nclass >> kready;
                t_parent = h_parent -> getPar(parentName);
                //cout << "Proverka2:  " << parentName << "  NEWPARENT:  " <<
t_parent -> getName() << "  CHILD:  " << childName << endl << endl;
                if (nclass == 2)
                        t_child = new Novel (t_parent, childName);
                else if (nclass == 3)
                        t_child = new Cl_2 (t_parent, childName);
                else if (nclass == 4)
                        t_child = new Cl_3 (t_parent, childName);
                else if (nclass == 5)
                        t_child = new Cl_4 (t_parent, childName);
                else if (nclass == 6)
                        t_child = new Cl_5 (t_parent, childName);
                t_child -> setReady(kready);
        }while (true);
```

```
}

int Appl :: exec_app () //Выполнение программы
{
        string str;
        cout << "Object tree\n" << this -> getName();
        this -> printTree (1);
        //this -> printStatus ()
        cin >> str;
        while (str != "//")
        {
                if (this -> getPar(str))
                {
                        cout << endl << str << " Object name: " << this ->
getPar(str) -> getName();
                        //cout << endl << "ADRES: " << this ->
objectWay(getPar(str)) << endl;
                }
                else
                        cout << endl << str << " Object not found";
                cin >> str;
        }
        return (0);
}
```

Файл Appl.h

```
#ifndef APPL_H
#define APPL_H

#include "Base.h"

class Appl : public Base
{
        public:
        Appl (Base* parent = 0);       //Конструктор
        void bild_tree_objects ();     //Построение дерева
        int exec_app ();               //Выполнение программы
};

#endif
```

Файл Base.cpp

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

```
Base :: Base (Base* parent, string name) //Конструктор
{
        setName (name);
        this -> parent = parent;
        if (parent)
                parent -> children.push_back (this);
}

void Base :: setName (string name) //Установление имени объекта
{
        this -> name = name;
}

string Base :: getName () //Возврат имени объекта
{
        return this -> name;
}

void Base :: switchParent (Base* newparent) //Поменять родителя объекта
{
        for (int i = 0; i < (this -> parent -> children.size()); i++)
//Поиск имени текущего объекта в
        {
                if ((this -> parent -> children[i] -> getName()) == this ->
getName())          //списке children его текущего родителя
                {
                        this -> parent -> children.erase(this -> parent ->
children.begin() + i);  //для удаления оттуда и замены его родителя
                        break;
                }
        }
        this -> parent = newparent;
        parent -> children.push_back(this);
}

Base* Base :: getParent () //Возврат родителя объекта
{
        return this -> parent;
}

void Base :: printNames () //Вывод имён объектов
{
        if (children.empty())
                return;
        cout << endl << name;
        it_children = children.begin();
        while (it_children != children.end())
        {
                cout << "  " << (*it_children) -> getName();
                it_children++;
        }
        it_children--;
        (*it_children) -> printNames();
}

Base* Base :: findParent (string parentName) //Нахождение указателя на объект
по имени
```

```
{
        Base* s_parent = 0;
        if (this -> getName() == parentName)
                return this;
        for (int i = 0; i < (this -> children.size()); i++)
        {
                if (this -> children[i] -> getName() == parentName)
                        return this -> children[i];
                else if ((s_parent) && s_parent -> getName() == parentName)
                        break;
                else
                        s_parent = this -> children[i] ->
findParent(parentName);
        }
        return s_parent;
}

void Base :: setReady (int k) //Установление готовности объекта
{
        this -> kready = k;
}

int Base :: statusReady() //Возврат готовности объекта
{
        return kready;
}

void Base :: printStatus ()  //Вывод статуса готовности
{
        /*cout << "Name: " << this -> getName() << endl << "<------" << endl;
        this -> printNames();
        cout << endl << "-------->" << endl;*/
        cout << endl << "The object " << this -> getName();
        if ( this -> statusReady() > 0)
                cout << " is ready";
        else
                cout << " is not ready";
        it_children = children.begin();
        while ( it_children != children.end() )
        {
                (*it_children) -> printStatus();
                it_children++;
        }
        it_children--;
}

void Base :: printTree (int level) //Вывод дерева иерархии
{
        string space;
        space.append ( 4 * level, ' ');
        it_children = children.begin();
        while ( it_children != children.end() )
        {
                cout << "\n" << space << (*it_children) -> getName();
                (*it_children) -> printTree(level + 1);
                it_children++;
        }
}

string Base :: objectWay (Base* t_parent) //Возврат пути до объекта
```

```
{
        if (!t_parent -> parent)
                return "/" + t_parent -> getName();
        return objectWay(t_parent -> parent) + "/" + t_parent -> getName();
}

Base* Base :: getPar (string parentName) //Возврат указателя на объект по его
пути
{
        if (parentName[0] == '/' && parentName[1] == '/')
                return findParent(parentName.substr(2, parentName.length() -
2));     return this -> findParent(findObj(parentName));

}

string Base :: findObj (string parentName) //Нахождение имени объекта по его
пути
{
        int x, y, r;
        Base* t_parent;
        if (parentName.find('/', 1) < parentName.length())
                t_parent = this -> findParent(parentName.substr(1,
parentName.find('/', 1) - 1));
        else
                t_parent = this -> findParent(parentName.substr(1,
parentName.find('/', 1)));
        parentName = parentName.substr(1, parentName.length() - 1);
        while (parentName.find('/') < parentName.length())
        {
                //cout << "STROKA: " << parentName << endl;
                r = -1;
                x = parentName.find('/', 1);
                y = parentName.find('/', x + 1);
                for (int i = 0; i < (t_parent -> children.size()); i++)
                {
                        if (t_parent -> children[i] -> getName() ==
parentName.substr(x + 1, parentName.length() - y - 1))
                                r = 1;
                }
                if (r == -1)
                        return "";
                t_parent = this -> findParent(parentName.substr(x + 1,
parentName.length() - y - 1));
                parentName = parentName.erase(0,x+1);
        }
        //cout << "Return: " << parentName << endl;
        return parentName;
}
```

Файл Base.h

```
#ifndef BASE_H
#define BASE_H
```

```
#include <vector>
#include <string>
using namespace std;
class Base
{
        string name;                                    //Имя объекта
        Base* parent;                                   //Указатель на родителя
объекта
объекта int kready;                                     //Статус готовности

        vector < Base* > children;                      //Список детей объекта
        vector < Base* > :: iterator it_children;       //Итератор для списка
детей объекта

        public:
        Base (Base* parent, string name = ""); //Конструктор
        void setName (string name);             //Установление имени объекта
        string getName ();                      //Возврат  имени  объекта
        void switchParent (Base* parent);       //Поменять родителя объекта
        Base* getParent ();                     //Возврат родителя объекта
        void printNames ();                     //Вывод имён объектов
        Base* findParent (string parentName);  //Нахождение указателя на
объект по имени
        void setReady (int k = 1);              //Установление готовности
объекта
        int statusReady ();                     //Возврат готовности объекта
        void printStatus ();                    //Вывод статуса готовности
        void printTree (int level);             //Вывод дерева иерархии
        string objectWay (Base* t_parent);      //Возврат пути до объекта
        Base* getPar (string parentName);       //Возврат указателя на объект
по его пути
        string findObj (string parentName);     //Нахождение имени объекта по
его пути
};

#endif
```

Файл Cl_2.cpp

```
#include "Cl_2.h"
using namespace std;
Cl_2 :: Cl_2 (Base* parent, string name) : Base (parent, name) {}
```

Файл Cl_2.h

```
#ifndef CL_2_H
```

```
#define CL_2_H
#include "Base.h"

class Cl_2 : public Base
{
        public:
        Cl_2(Base*, string);
};

#endif
```

Файл Cl_3.cpp

```
#include "Cl_3.h"
using namespace std;
Cl_3 :: Cl_3 (Base* parent, string name) : Base (parent, name) {}
```

Файл Cl_3.h

```
#ifndef CL_3_H
#define CL_3_H
#include "Base.h"

class Cl_3 : public Base
{
        public:
        Cl_3(Base*, string);
};

#endif
```

Файл Cl_4.cpp

```
#include "Cl_4.h"
using namespace std;
Cl_4 :: Cl_4 (Base* parent, string name) : Base (parent, name) {}
```

Файл Cl_4.h

```
#ifndef CL_4_H
#define CL_4_H
#include "Base.h"

class Cl_4 : public Base
{
        public:
        Cl_4(Base*, string);
};

#endif
```

Файл Cl_5.cpp

```
#include "Cl_5.h"
using namespace std;
Cl_5 :: Cl_5 (Base* parent, string name) : Base (parent, name) {}
```

Файл Cl_5.h

```
#ifndef CL_5_H
#define CL_5_H
#include "Base.h"

class Cl_5 : public Base
{
        public:
        Cl_5(Base*, string);
};

#endif
```

Файл main.cpp

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

int main()
{
```

```
        Appl ob_app;
        ob_app.bild_tree_objects ();    // построение дерева объектов
        return ob_app.exec_app (); // запуск системы
        //return(0);
}
```

Файл Novel.cpp

```
#include "Novel.h"
//#include <string>
using namespace std;
Novel :: Novel (Base* parent, string name) : Base (parent, name) {}
```

Файл Novel.h

```
#ifndef NOVEL_H
#define NOVEL_H
#include "Base.h"

class Novel : public Base
{
        public:
        Novel(Base*, string);
};

#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root /root object_1 3 1 /root object_2 2 1 /root/object_2 object_4 3 -1 /root/object_2 object_5 4 1 /root object_3 3 1 /root/object_2 object_3 6 1 /root/object_1 object_7 5 1 /root/object_2/object_4 object_7 3 -1 endtree /root/object_1/object_7 //object_7 /root/object_1/object_3 //	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 /root/object_1/object_7 Object name: object_7 //object_7 Object name: object_7 /root/object_1/object_3 Object not found	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 /root/object_1/object_7 Object name: object_7 //object_7 Object name: object_7 /root/object_1/object_3 Object not found

ЗАКЛЮЧЕНИЕ

Благодаря данным работам я научился разрабатывать базовые объектно-ориентированные программы на языке C++ и использовать основные библиотеки языка для разработки данных программ. Также мне это помогло понять как разрабатывать программы с использованием наследования классов и разрабатывать базовый класс для объектов. Помимо всего прочего я разобрался как определять общий функционал для используемых в рамках приложения объектов и разрабатывать операции добавления, удаления, изменения позиции объекта в рамках иерархического дерева, а также разрабатывать алгоритм для вывода дерева иерархии объектов по уровням. Кроме того я научился построению дерева иерархии объектов и освоил алгоритмы обработки структур данных в виде дерева. Кроме того я освоил знания по переключению состояния объектов и определению их готовности к работе, и по поиску указателя на объект по координате по дереву иерархии объектов или имени, при уникальности наименований объектов.

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).