МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« КЛ_3_3 Определение указателя на объект по его координате »

С тудент группы	ИНБО-15-20	Ло В.Х.
Руководитель практики	Ассистент	Рогонова О.Н.
Работа представлена	«___»_____2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1.Почему ООП актуален и будет актуальным? (ООП - отображение окружающего нас объектного мира……)

ООП стало неотемлемой частью разработки многих современных проектов. Считается, что ООП повышает производительность, упрощает обслуживание и расширяет программое обеспечение, позволяя программистам сосредоточиться на программых объектах более высокого порядка. ООП старается спроецировать объекты реального мира и бизнес процессов в программый код. Эта парадигмы старается соденить поведение сущности с ее данным. На основе классов создается множесто переменных, объекты получают в свое распоряжение индивидуальное пространства имен. Наследование же позволяет не писать новый код под конкретные данные, а использовать уже сушествующий. И многие программисты уже начали использовать эту парамигму в своих проектах. Именно поэтому ООП будет актуально еще дольгое время.

2.Почему C++ наиболее удобен для изучения ООП?

C++ хорошо подходит для изучение ООП потому что тут относительно чистая реализация ООП без всяких синтаксических излишеств. Четко разгранниченный уровни доступа к членам класса, возможность множественного начледования и динамических полиморфизм дают возможность быстро усвоить основые концепции ООП. В дальнейшем, основываясь на полученных знаниях можно легко понять весь синффывцытаксис и в других языках.

3.Что дает ООП на C++ для Вашего направления?

ООП позваляет более структуризированно управлять большими данными.

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

Потоки ввод/вывод cin/cout.

1.Объект класса: cl_base

Описание класса: cl_base

Свойства:

+наименование объекта: строкового типа

+список указателей на объекты плеченных к текущему объекту в дереве иерахии

Функционал:

-параметриванный конструктор с параметрами

-Определния имени объекта

-получения имени объекта

-получения указателя на головной объект текущего объекта

-толкнуть объектов в конец векторов

-выбирять объектов в дереве иерахии

-установить состояния объекта

получить состояния объектов

2.Описание класса: cl_application наследуется от cl_base

Методы:

bild_tree_objects()-постоить медель иерахической системы

exec_app()-Реализовать вывод на консоль иерахического дерева объектов

№	Имя класса	класса следник	Модификатор доступа при наследовании	Описание	номер
1				Базовый класса в иерахии классов . Содержит основный поля и методы	
	cl-base	cl_application	public		2
		cl_2	public		
		cl_3	public		
		cl_4	public		
		cl_5	public		
		cl_6	public		
2	cl_application			Класс корневого объектв	

Описание алгоритма

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

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base()

Функционал: параметризиванные параметры

Параметры: cl_base* parent,string name

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода cl_base() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		параметризиванные параметры	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_object_name()

Функционал: Определния имени объекта

Параметры: name

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода set_object_name() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		Определния имени объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_object_name()

Функционал: получения имени объекта

Параметры: нет

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода get_object_name() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		получения имени объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_parent()

Функционал: переопределение объекта для текущего в дереве иерахии

Параметры: cl_base* parent

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода set_parent() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		переопределение объекта для текущего в дереве иерахии	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_parent()

Функционал: получения указателя на головной объект текущего объекта

Параметры: нет

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода get_parent() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		получения указателя на головной объект текущего объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_status

Функционал: вводит номер исходного состояния очередного объекта

Параметры: status

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода set_status класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		вводит номер исходного состояния очередного объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_by_name()

Функционал: находим элемент с именем

Параметры: name

Возвращаемое значение: nullptr

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода get_by_name() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		находим элемент с именем	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: show_tree_view()

Функционал: вводит информацию о вершинах в виде дерева

Параметры: deep_lvl

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода show_tree_view() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		вводит информацию о вершинах в виде дерева	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_by_path()

Функционал: В качестве параметра методу передать путь обекта от корневого

Параметры: path

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода get_by_path() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		string name_now = "",temp	2	
2		int i = 1	3	
3	i<path.size()		4	
			6	
4	path[i]!= '/'	name_now+=path[i];	5	
		break	5	
5		i++	3	
6		temp=path;	7	

		path="";		
7		temp=path; path=""; int i=name_now.size()+1	8	
8	i<temp.size()	path +=temp[i]	9	
			10	
9		i++	8	
10	path==""&&this->name==name_now	return this	11	
			11	
11	!children.size()	return 0	12	
		name_now=""	12	
12		int i=1	13	
13	i<path.size()		14	
			16	
14	path[i]!='/'	name_now+=path[i]	15	
		break	17	
15		i++	13	
16		int i=0	17	
17	i<children.size()		18	
			20	
18	children[i]->get_object_name()==name_now	return children[i]->get_by_path(path)	19	
			20	
19		i++	17	
20		return 0	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: bild_tree_objects()

Функционал: построение дерева объекта

Параметры: нет

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 11.

Таблица 11. Алгоритм метода bild_tree_objects() класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		cl_2* ob_2; cl_3* ob_3; cl_4* ob_4; cl_5* ob_5; cl_6* ob_6; string nameParent,nameChild; int class_type, status=1;	2	
2		cin>>nameParent	3	
3		this->set_object_name(nameParent)	4	
4	true	cin>>nameParent	5	
			∅	
5	nameParent=="endtree"	break	∅	
			6	
6		cin>>nameChild>>class_type>>status	7	
7		cl_base* b=this->get_by_path(nameParent)	8	
8	class_type==2	ob_2=new cl_2(b,nameChild); ob_2->set_status(status	∅	
			9	
9	class_type==3	ob_3=new cl_3(b,nameChild); ob_3->set_status(status	∅	

			10	
10	class_type==4	ob_4=new cl_4(b,nameChild); ob_4->set_status(status	∅	
			11	
11	class_type==5	ob_5=new cl_5(b,nameChild); ob_5->set_status(status	∅	
			12	
12	class_type==6	ob_6=new cl_6(b,nameChild); ob_6->set_status(status	4	
			4	

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app()

Функционал: Реализовать вывод на консоль иерахического дерева объектов

Параметры: нет

Возвращаемое значение: int код, возврата

Алгоритм метода представлен в таблице 12.

Таблица 12. Алгоритм метода exec_app() класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		cout<<"Object tree"<<endl	2	
2		this->show_tree_view(0)	3	
3		cin>>way	4	
4	way!="//"		5	
			10	

5	way[0]=='/'&&way[1]=='/'	string tmp=""; int i=2	6	
			9	
6	i<way.size()	tmp+=way[i]	7	
			8	
7		i++	6	
8	get_by_name(tmp)	cout<<endl<<way<<" Object name:"<<get_by_name(tmp)->get_object_name();	10	
		cout<<endl<<way<<" Object not found"	10	
9	get_by_path(way)	cout<<endl<<way<<" Object name:"<<get_by_path(way)->get_object_name();	10	
		cout<<endl<<way<<" Object not found"	10	
10		cin>>way;	11	
11		return 0	∅	

Функция: main

Функционал: основнная программа

Параметры: нет

Возвращаемое значение: int код,возврата

Алгоритм функции представлен в таблице 13.

Таблица 13. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		cl_application ob_cl_application	2	
2		ob_cl_application.bild_tree_objects()	3	
3		return ob_cl_application.exec_app()	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

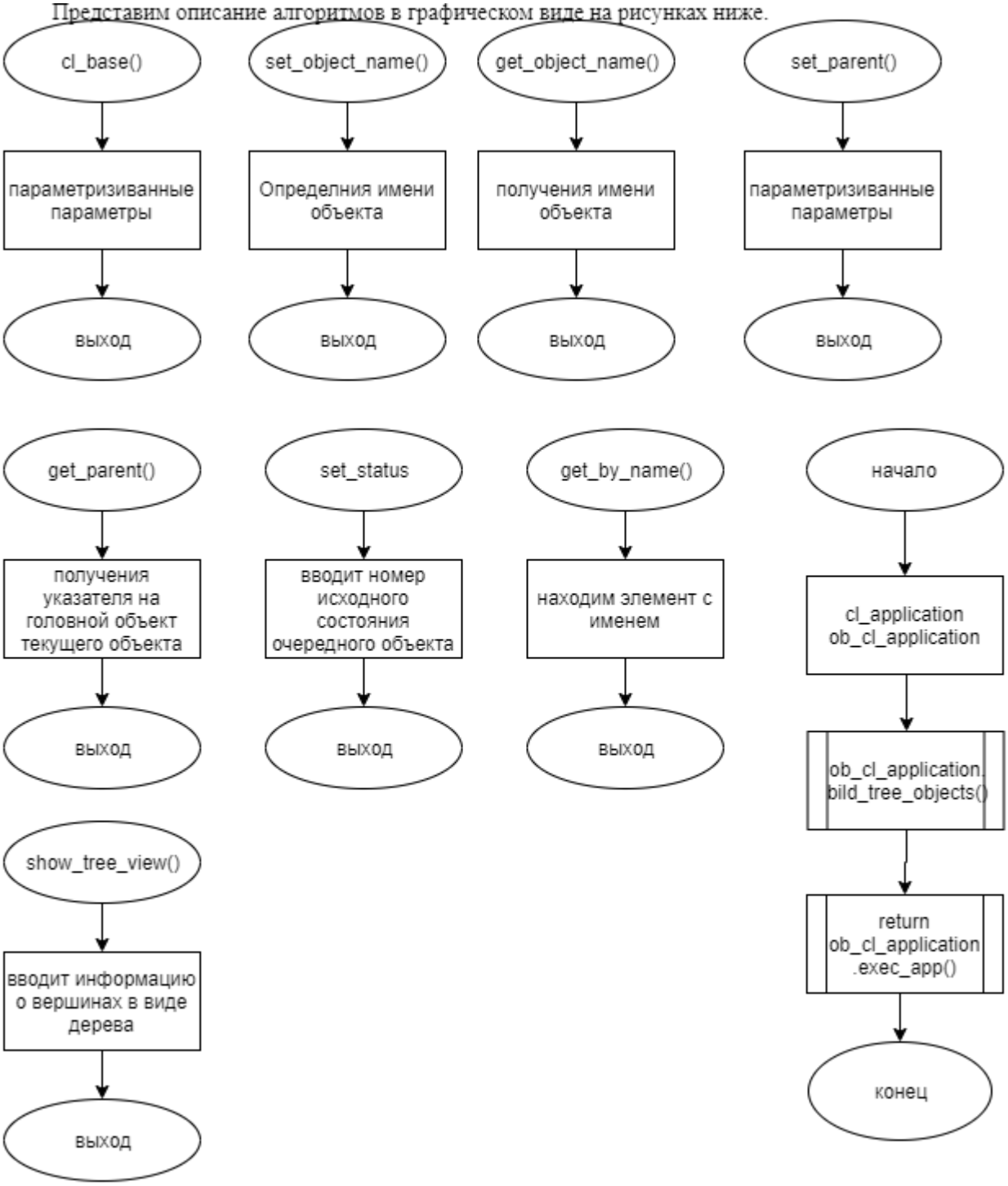

Рис. 1. Блок-схема алгоритма.

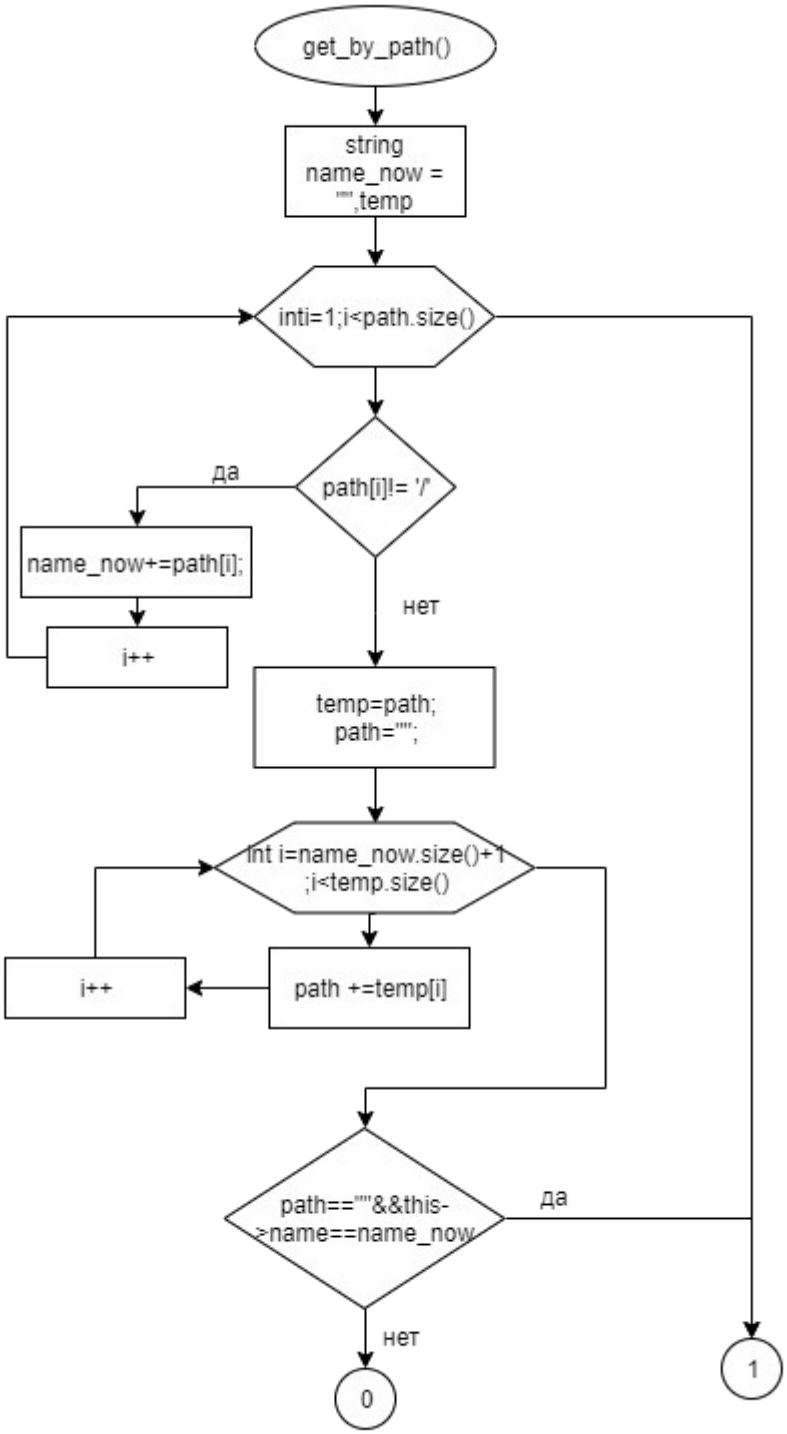

Рис. 2. Блок-схема алгоритма.

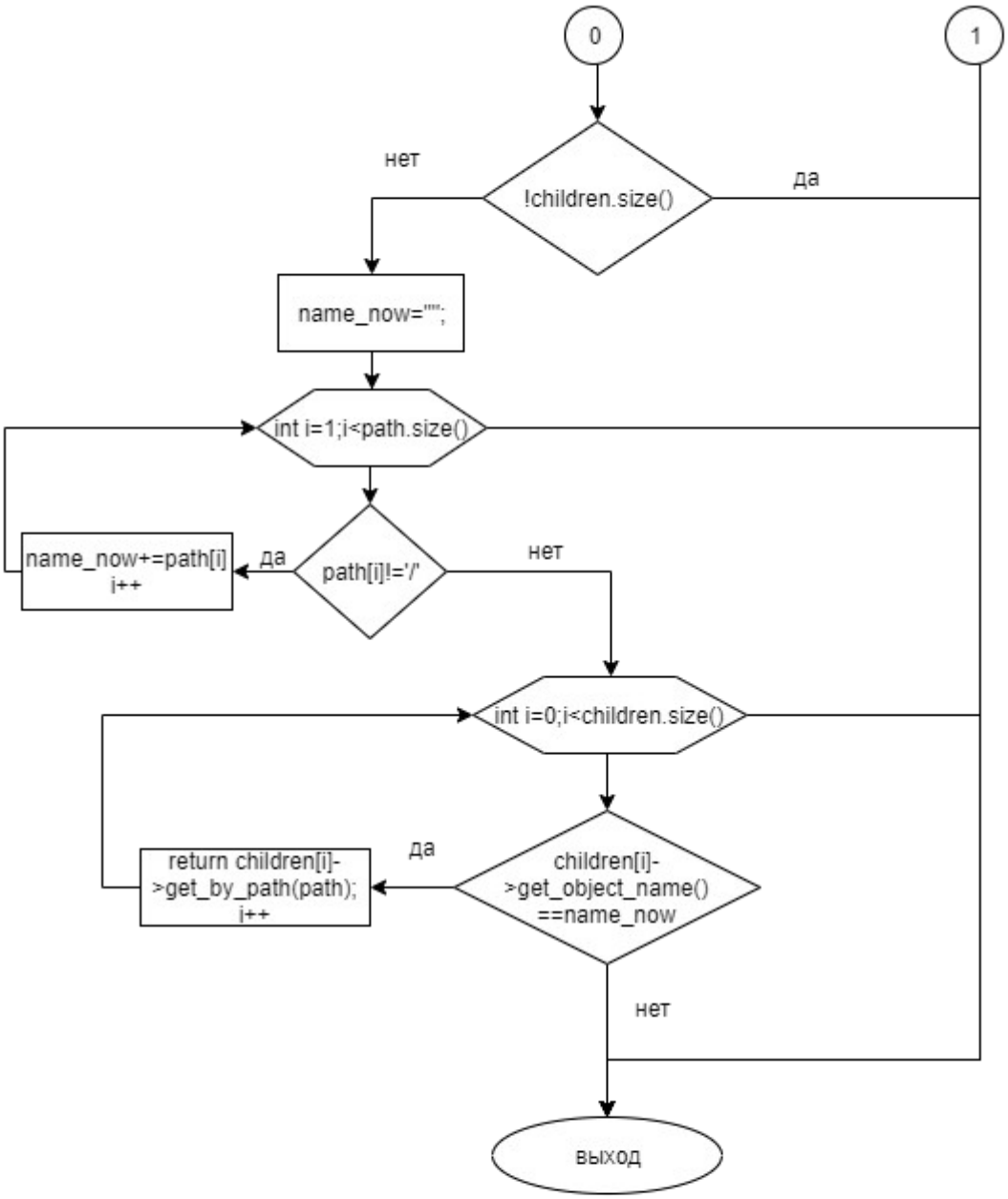

Рис. 3. Блок-схема алгоритма.

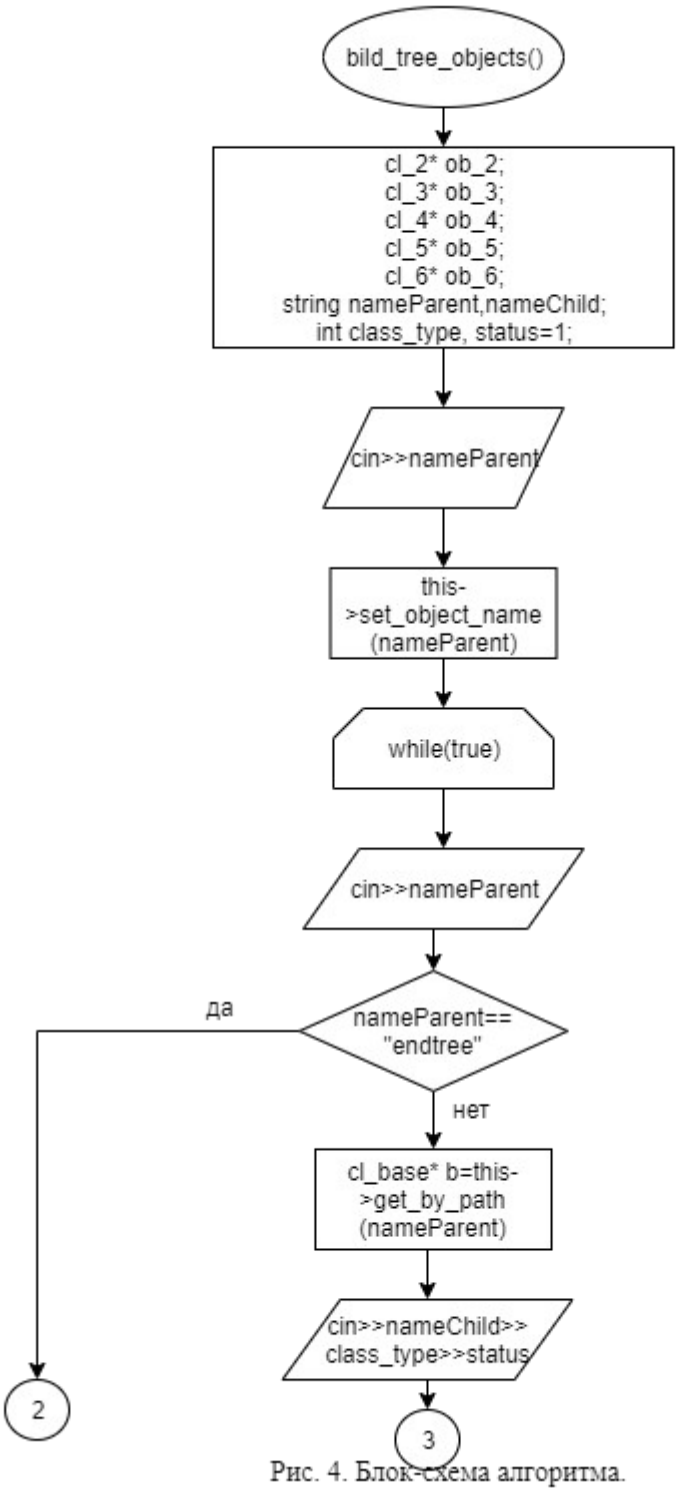

Рис. 4. Блок-схема алгоритма.

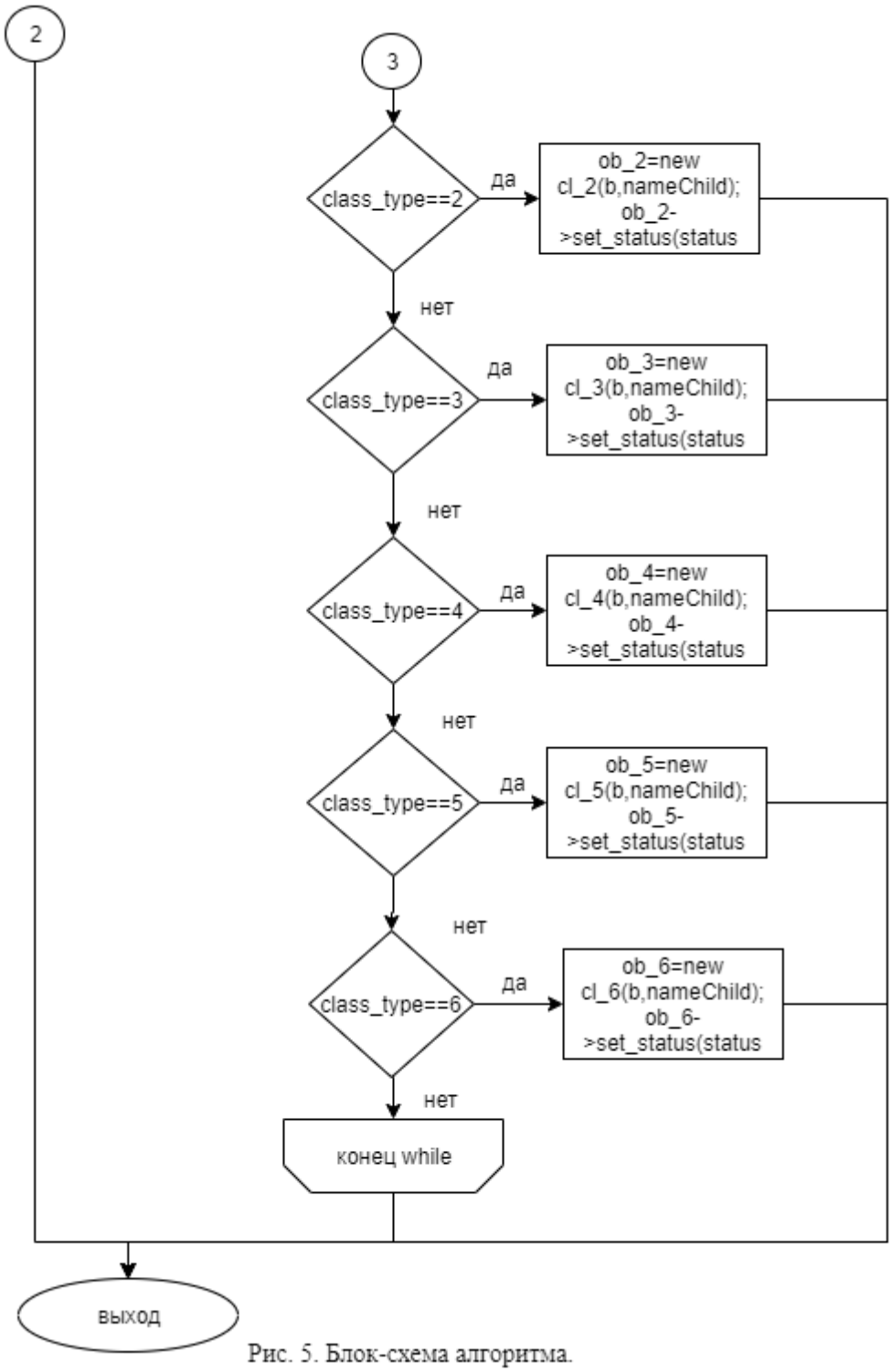

Рис. 5. Блок-схема алгоритма.

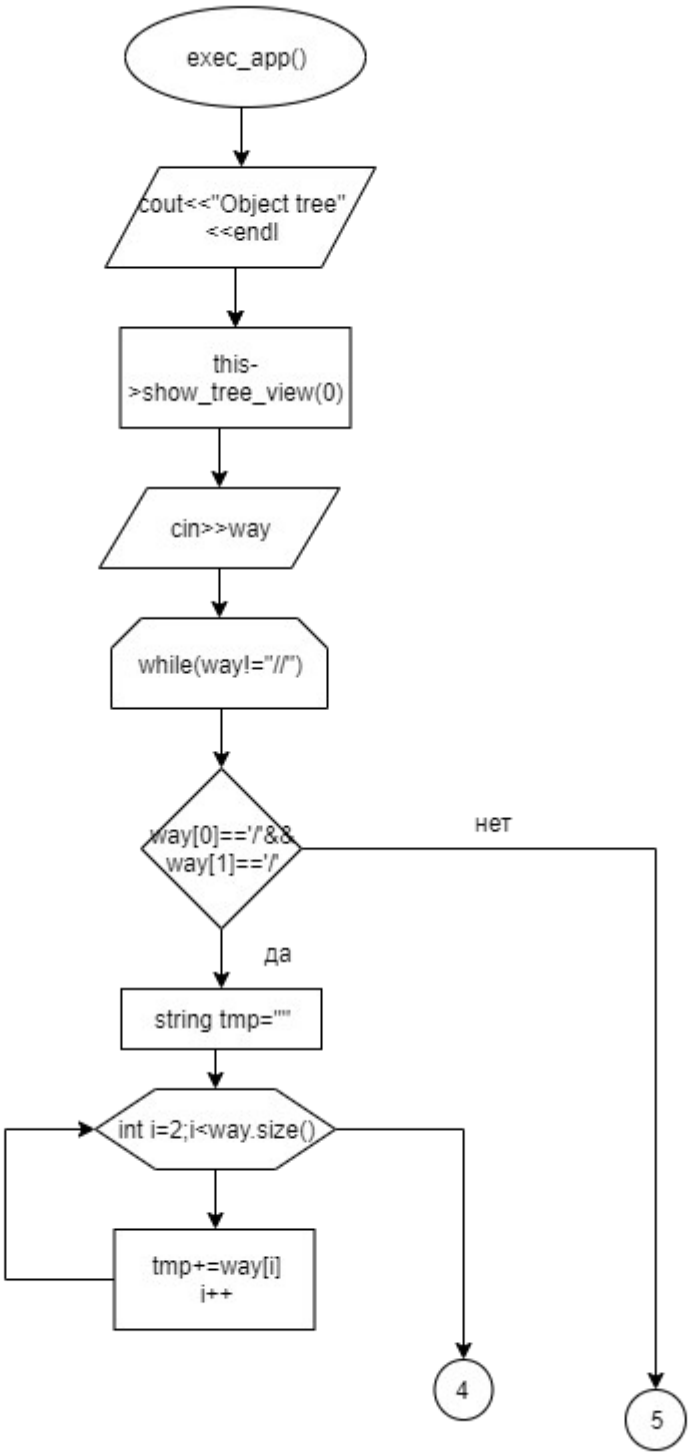

Рис. 6. Блок-схема алгоритма.

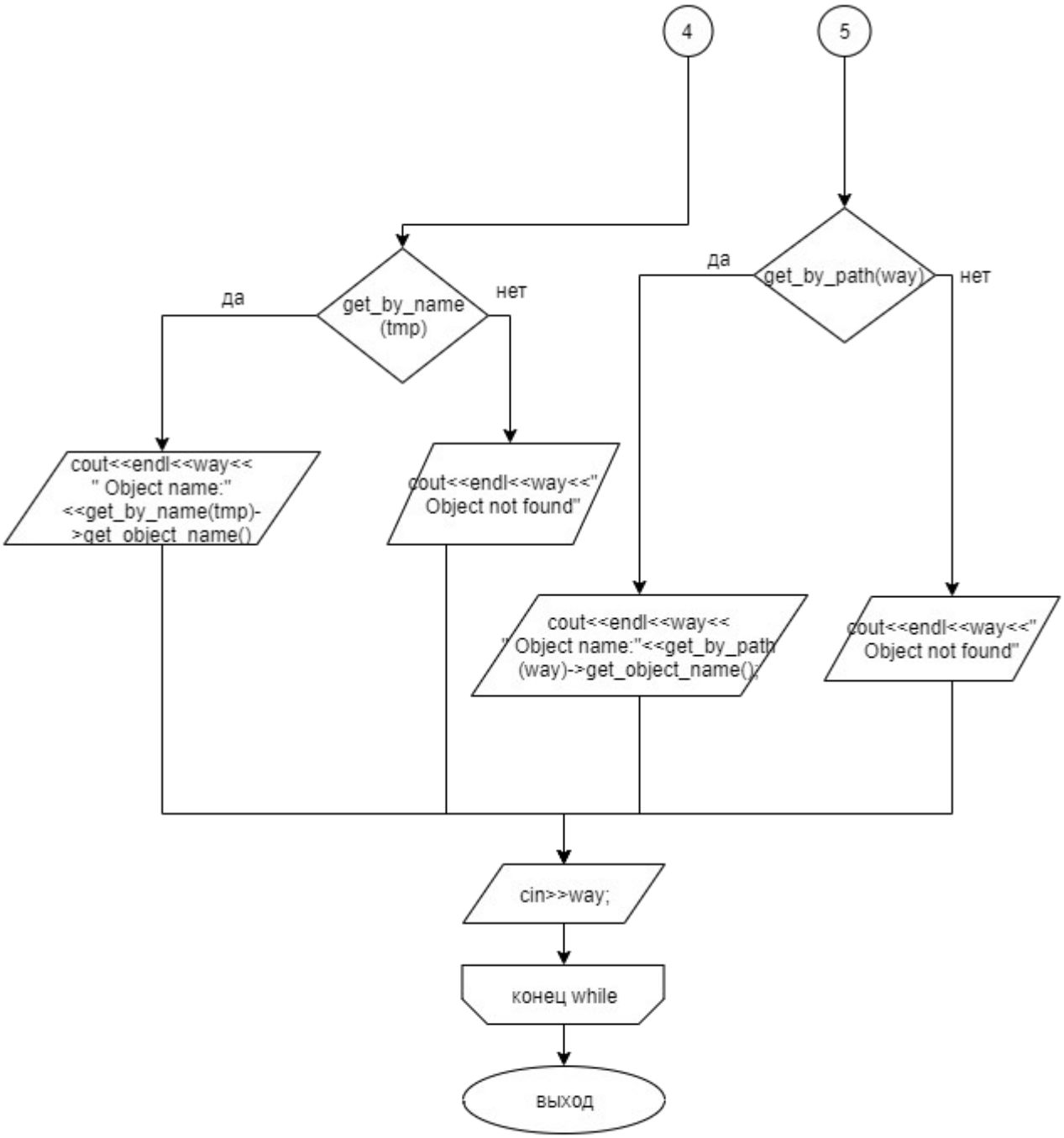

Рис. 7. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл cl_2.h

```
#ifndef CL_2_H
#define CL_2_H
#include<string>
#include "cl_base.h"
class cl_2 : public cl_base
{
public:
                cl_2(cl_base* parent,string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_3.h

```
#ifndef CL_3_H
#define CL_3_H
#include<string>
#include "cl_base.h"
class cl_3 : public cl_base
{
public:
                cl_3(cl_base* parent,string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_4.h

```
#ifndef CL_4_H
#define CL_4_H
#include<string>
#include "cl_base.h"
class cl_4 : public cl_base
{
public:
                cl_4(cl_base* parent,string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_5.h

```
#ifndef CL_5_H
#define CL_5_H
#include<string>
#include "cl_base.h"
class cl_5 : public cl_base
{
public:
                cl_5(cl_base* parent,string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_6.h

```
#ifndef CL_6_H
#define CL_6_H
#include<string>
#include "cl_base.h"
class cl_6 : public cl_base
{
public:
                cl_6(cl_base* parent,string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_application.cpp

```
#include<iostream>
#include "cl_application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include "cl_base.h"
using namespace std;
void cl_application::bild_tree_objects()
{
        cl_2* ob_2;
        cl_3* ob_3;
        cl_4* ob_4;
        cl_5* ob_5;
        cl_6* ob_6;
        string nameParent,nameChild;
        int class_type, status=1;
```

```
        cin>>nameParent;
        this->set_object_name(nameParent);
        while(true)
        {
                cin>>nameParent;
                if(nameParent=="endtree")
                {
                        break;
                }
                cin>>nameChild>>class_type>>status;
                cl_base* b=this->get_by_path(nameParent);
                if(class_type==2)
                {
                        ob_2=new cl_2(b,nameChild);
                        ob_2->set_status(status);
                }
                if(class_type==3)
                {
                        ob_3=new cl_3(b,nameChild);
                        ob_3->set_status(status);
                }
                if(class_type==4)
                {
                        ob_4=new cl_4(b,nameChild);
                        ob_4->set_status(status);
                }
                if(class_type==5)
                {
                        ob_5=new cl_5(b,nameChild);
                        ob_5->set_status(status);
                }
                if(class_type==6)
                {
                        ob_6=new cl_6(b,nameChild);
                        ob_6->set_status(status);
                }
        }
}
int cl_application::exec_app()
{
        cout<<"Object tree"<<endl;
        this->show_tree_view(0);
        string way;
        cin>>way;
        while(way!="//")
        {
                if(way[0]=='/'&&way[1]=='/')
                {
                        string tmp="";
                        for(int i=2;i<way.size();i++)
                        {
                                tmp+=way[i];
                        }
                        if(get_by_name(tmp))
                                cout<<endl<<way<<" Object name:
"<<get_by_name(tmp)->get_object_name();
                        else
                                cout<<endl<<way<<" Object not found";
                }
                else
```

```
                {
                        if(get_by_path(way))
                                cout<<endl<<way<<" Object name:
"<<get_by_path(way)->get_object_name();
                        else
                                cout<<endl<<way<<" Object not found";
                }
                cin>>way;
        }
        return 0;
}
```

Файл cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H
#include "cl_base.h"
class cl_application : public cl_base
{
        public:
                        void bild_tree_objects();
                        int exec_app();
};
#endif
```

Файл cl_base.cpp

```
#include<iostream>
#include "cl_base.h"
cl_base::cl_base(cl_base* parent,string name)
{
        set_object_name(name);
        this->parent = parent;
        if(this->parent != 0)
        {
                this->parent->children.push_back(this);
}
}
cl_base::~cl_base()
{
        for(int i=0;i<this->children.size();i++)
        {
                delete this->children[i];
        }
}
void cl_base::set_object_name (string name)
{
        this->name = name;
}
```

```
string cl_base::get_object_name()
{
        return this->name;
}

void cl_base::set_parent(cl_base* parent)
{
        if(parent)
        {
                if(this->parent)
                {
                        for(int i = 0 ; i<this->parent->children.size();i++)
                        {
                                if(this->parent->children[i] == this)
                                {
                                        this->parent-
>children.erase(children.begin()+i);
                                }
                        }
                }
                this->parent = parent;
                this->parent->children.push_back(this);
        }
}
cl_base* cl_base:: get_parent()
{
        return this->parent;
}
void cl_base::show_tree()
{
        if(!this->children.size())
                return;
                cout<<this->name;
        for(int i=0;i < this->children.size();++i)
        {
                cout<<"  ";
                cout<<this->children[i]->get_object_name();
        }
        bool w=false;
        for(int i=0;i<this->children.size();++i)
        {
                if(this->children[i]->children.size() && !w)
                {
                        cout<<"\n";
                        w=true;
                }
                this->children[i]->show_tree();
        }
        return;
}
void cl_base::set_status(int status)
{
this->status=status;
}
void cl_base::show_status_tree()
{
        cout<<"The object "<<this->name;
        if(this->status>0){
                cout<<" is ready";
        }
```

```
        else
        {
                cout<<" is not ready";
        }
        if(this->children.size())
        {
                cout<<"\n";
        }
        for(int i=0;i<this->children.size();++i)
        {
                this->children[i]->show_status_tree();
                if(i!=this->children.size()-1)
                        cout<<endl;
        }
        return;
}
void cl_base::show_tree_view(int deep_lvl=0)
{
        int indent = deep_lvl*4;
        for(int i=0;i<indent;i++)
        {
                cout<<" ";
        }
        cout<<this->name;
        if(!this->children.size())
                return;
        cout<<endl;
        for(int i = 0;i<this->children.size();i++)
        {
                this->children[i]->show_tree_view(deep_lvl +1);
                if(i!= this->children.size()-1)
                {
                        cout<<endl;
                }
        }
        return;
}
cl_base* cl_base::get_by_name(string name)
{
        if(this->name==name){
                return this;
        }
        for(int i=0;i<this->children.size();++i)
        {
                cl_base* b=children[i]->get_by_name(name);
                if(b!=NULL)
                        return b;
        }
        return nullptr;
}
cl_base* cl_base::get_by_path(string path)
{
string name_now = "",temp;
for(int i = 1;i<path.size();i++)
{
        if(path[i]!= '/')
                name_now+=path[i];
```

```
        else
                break;
}
        temp=path;
        path="";
        for(int i=name_now.size()+1;i<temp.size();i++)
        {
                path +=temp[i];
        }
        if(path==""&&this->name==name_now)
                return this;
        if(!children.size())
                return 0;
        else
        {
                name_now="";
                for(int i=1;i<path.size();i++)
                {
                        if(path[i]!='/')
                                name_now+=path[i];
                        else
                                break;
                }
                for(int i=0;i<children.size();i++)
                {
                        if(children[i]->get_object_name()==name_now)
                                return children[i]->get_by_path(path);
                }
        }
        return 0;
}
```

Файл cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H
#include<iostream>
#include<vector>
#include<string>
using namespace std;
class cl_base
{
        private:
                        string name;
                        cl_base* parent;
                        int status;
                        vector<cl_base*> children;
        public:
                        cl_base(cl_base* parent=nullptr,string name=
"cl_base");
                        ~cl_base();
                        void set_object_name (string name);
                        string get_object_name();
                        void set_parent(cl_base* parent);
                        cl_base* get_parent();
                        void show_tree();
```

```
                        void show_status_tree();
                        cl_base* get_by_name(string name);
                        void set_status(int status);
                        void show_tree_view(int deep_lvl);
                        cl_base* get_by_path(string path);
};
#endif
```

Файл main.cpp

```
#include "cl_application.h"
int main()
{
                cl_application ob_cl_application;
                ob_cl_application.bild_tree_objects();
                return ob_cl_application.exec_app();
}
```

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 //object_1 /root/object_2/object_4 //object_3 /root/object_7 //	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 //object_1 Object name: object_1 /root/object_2/object_4 Object name: object_4 //object_3 Object name: object_3 /root/object_7 Object not found	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 //object_1 Object name: object_1 /root/object_2/object_4 Object name: object_4 //object_3 Object name: object_3 /root/object_7 Object not found

ЗАКЛЮЧЕНИЕ

В процесс выполнения курсовой работы Я научился:

- разрабатывать базовый класс для объектов;
- определять общий функционал для используемых в рамках приложения объектов;
- построению дерева иерархии объектов;
- освоил алгоритмы обработки структур данных в виде дерева;
- переключению состояния объектов и определению их готовности к работе;
- выводить на печать дерева иерархии объектов;
- искать указатель на объект по координате по дереву иерархии объектов или по 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).