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	ИКБО-04-20	Тарураев Е.А.
Руководитель практики	Ассистент	Данилович Е.С.
Работа представлена	«___»_____2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объектно-ориентированное программирование - парадигма программирования, пришедшая на замену функциональному подходу и представляющая из себя дробление задачи на отдельные объекты, между которыми и происходит взаимодействие. Основные концепции объектно-ориентированного программирования - это понятия объектов и классов. Главным понятием ООП является объект.

Объект - это имеющий название и особое поведение экземпляр класса. Поведение объекта зависит от методов, описанных внутри класса.

На основе классов может быть создано множество объектов, каждый из которых может обладать собственными значениями полей. Свойства объектов имеют одинаковые в каждом классе, но это не мешает получать доступ к их свойствам через объект, так как у каждого объект есть своё пространство имен.

C++ это удобный язык для изучения ООП, так как обладает следующими преимуществами:

- уровни доступа к элементам класса строго разграничены
- обладает полноценной реализацией ООП
- есть возможность реализовать множественное наследование
- есть реализация динамического полиморфизма
- возможность самостоятельно контролировать ресурсы памяти и очщать динамически выделенные ячейки пмяти, что способствует более глубокому пониманию ООП
- возможность быстрого изучения основных концепций ООП(абстракция,

наследование, инкапсуляция и полиморфизм

Таким образом ООП служит хорошим помощником в изучении программирования.

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

Иерархия наследования отображена в таблице 1.

Таблица 1. "Описание иерархии наследования классов"

№	Имя класса	Классы наследования	Модификатор доступа при наследовани	Описание	Номер
1	cl_base			Базовый класс	
		cl_application	public		2
		mikky2	public		3
		mikky2	public		4
		mikky4	public		5
		mikky5	public		6
		mikky6	public		7
2	cl_application			Класс-приложение	
3	mikky2			Класс подчиненного объекта	
4	mikky3			Класс подчиненного объекта	
5	mikky4			Класс подчиненного объекта	
6	mikky5			Класс подчиненного объекта	
7	mikky6				

cl_base - базовый класс

Методы:

- find_ob_koord

cl_application - класс-приложение, наследник cl_base

методы:

- bild_tree_objects
- exec_app

Классы mikky2, mikky3, mikky4, mikky5, mikky6 - наследники класса cl_base

Методы:

- Конструктор, передаёт параметры конструктору базового класса

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: find_ob_koord

Функционал: ищет адрес объекта

Параметры: string name, координаты объекта

Возвращаемое значение: cl_base*, адрес нужного объекта

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода find_ob_koord класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1	второй символ строки name это '/'	возврат метода find_ob с именем искомого объекта в качестве параметра	∅	
			2	
2		инициализация строковой переменной string значением пустой строки	3	
3		посимвольное копирование в переменную temp первого слова из name	4	
4	temp не равен	возврат NULL	∅	

	имени корня			
			5	
5		стереть первое слово из name	6	
6	name пуст	возврат this	∅	
			7	
7		temp равен пустой строке	8	
8		посимвольное копирование в переменную temp второго слова из name	9	
9		установка итератора на начало списка наследников	10	
10	список наследников обработан не полностью		11	
			13	
11	temp равен имени ребенка, на которого указывает итератор	возврат метода find_ob_koord(name) для этого ребенка	∅	
			12	
12		итератор переходит на следующее значение вектора children	10	
13		возврат NULL	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app

Функционал: запуск программы

Параметры: нет

Возвращаемое значение: int, код возврата

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		вывод "Object tree\n" вывод результат метода get_object_name	2	
2		вызов метода output_stairs() с параметром 1	3	
3		объявление строковой переменной coordy	4	
4		ввод coordy	5	
5	coordy не равен "//"		6	
			8	
6	find_ob_koord(coordy) вернул не NULL	вывод coordy," Object name ", имя объекта	7	
		вывод coordy, "Object not found "	7	
7		ввод coordy	5	
8		возврат 0	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: bild_tree_objects

Функционал: строит дерево объектов

Параметры: нет

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода bild_tree_objects класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		объявление переменных строкового типа imya_first, imya_glav, imya_current объявление переменных целочисленного типа num_of_class, ready	2	
2		ввод imya_first	3	
3		вызов метода set_object_name() с параметром imya_first вызов метода set_cond() с парметром 1 инициализация ppp типа cl_base* значением this инициализация prob типа cl_base* значением nullptr	4	
4		ввод imya_glav	5	
5	imya_glav равно "endtree"	выход	∅	
		ввод imya_current, nomer_class, gotov	6	
6		присваиваем ppp значение find_ob_koord(imya_glav)	7	
7	ppp равно 0		5	
			8	
8	nomer_class равно 2	создаём новый объект prob класса mikky2()с параметрами конструктора ppp, imya_current	4	

		вызов метода set_gotov() с параметром gotov		
	nomer_class равно 3	создаём новый объект prob класса mikky3()с параметрами конструктора ppp, imya_current вызов метода set_gotov() с параметром gotov	4	
	nomer_class равно 4	создаём новый объект prob класса mikky4()с параметрами конструктора ppp, imya_current вызов метода set_gotov() с параметром gotov	4	
	nomer_class равно 5	создаём новый объект prob класса mikky5()с параметрами конструктора ppp, imya_current вызов метода set_gotov() с параметром gotov	4	
	nomer_class равно 6	создаём новый объект prob класса mikky6()с параметрами конструктора ppp, imya_current вызов метода set_gotov() с параметром gotov	4	
			4	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

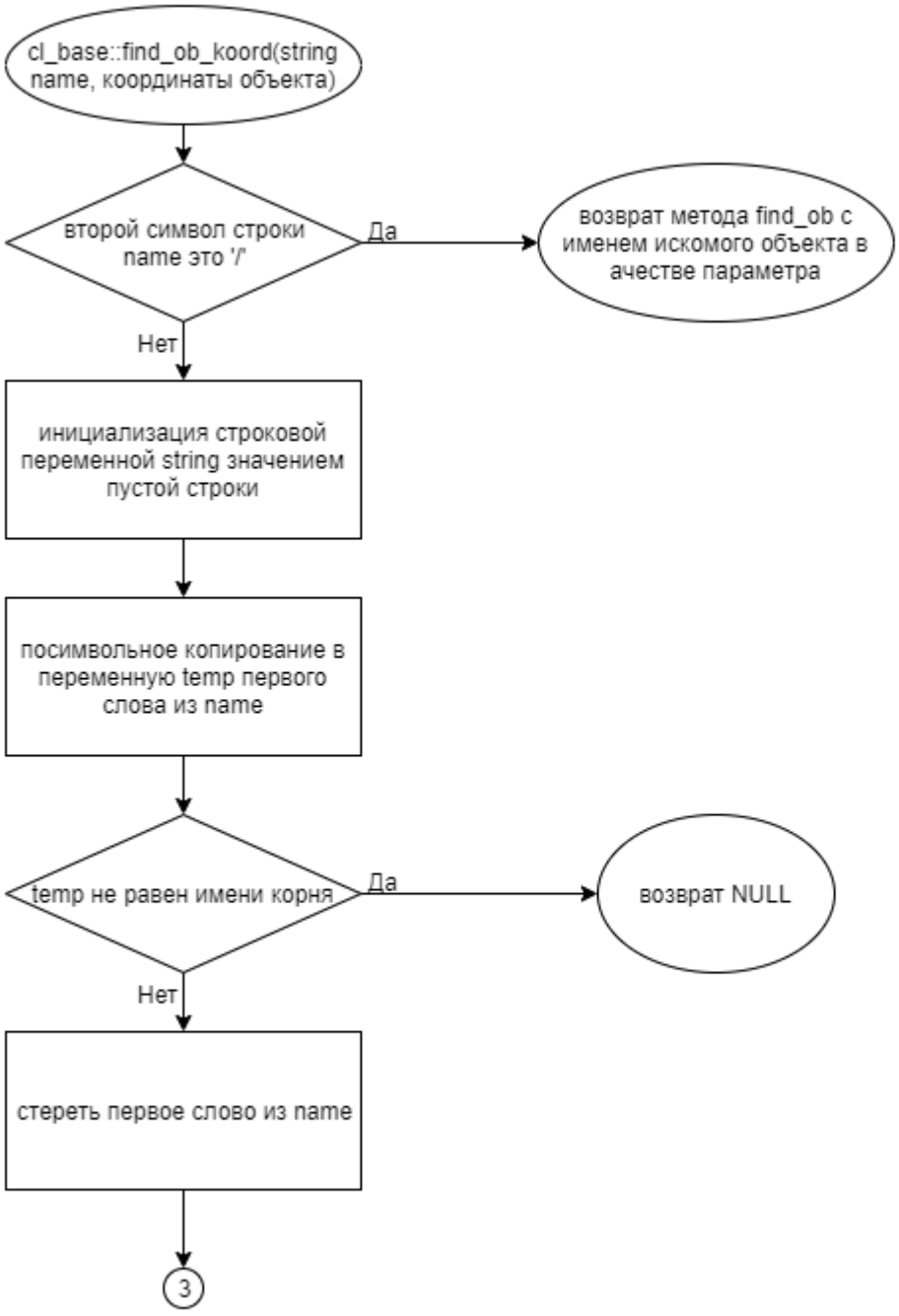

Рис. 1. Блок-схема алгоритма.

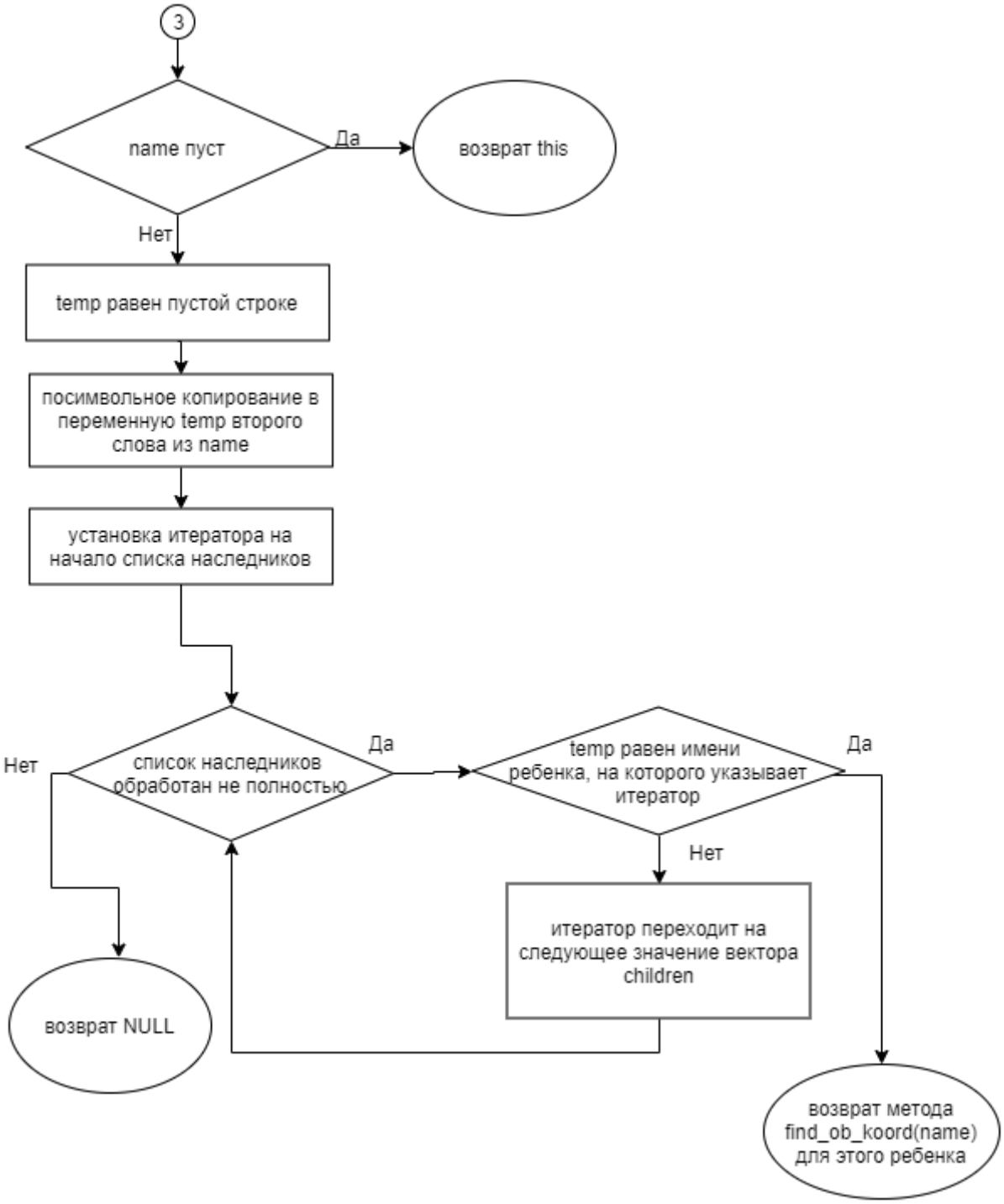

Рис. 2. Блок-схема алгоритма.

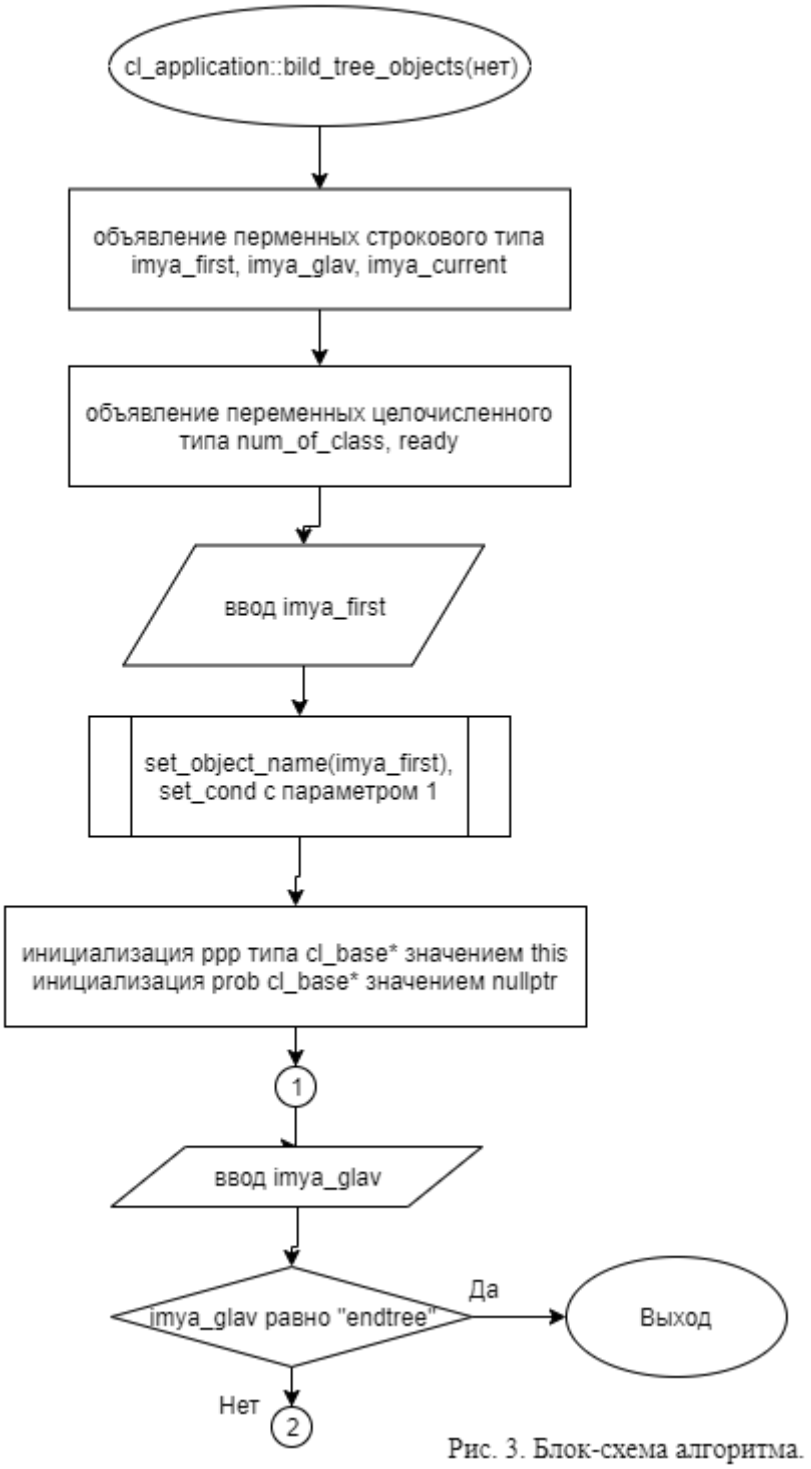

Рис. 3. Блок-схема алгоритма.

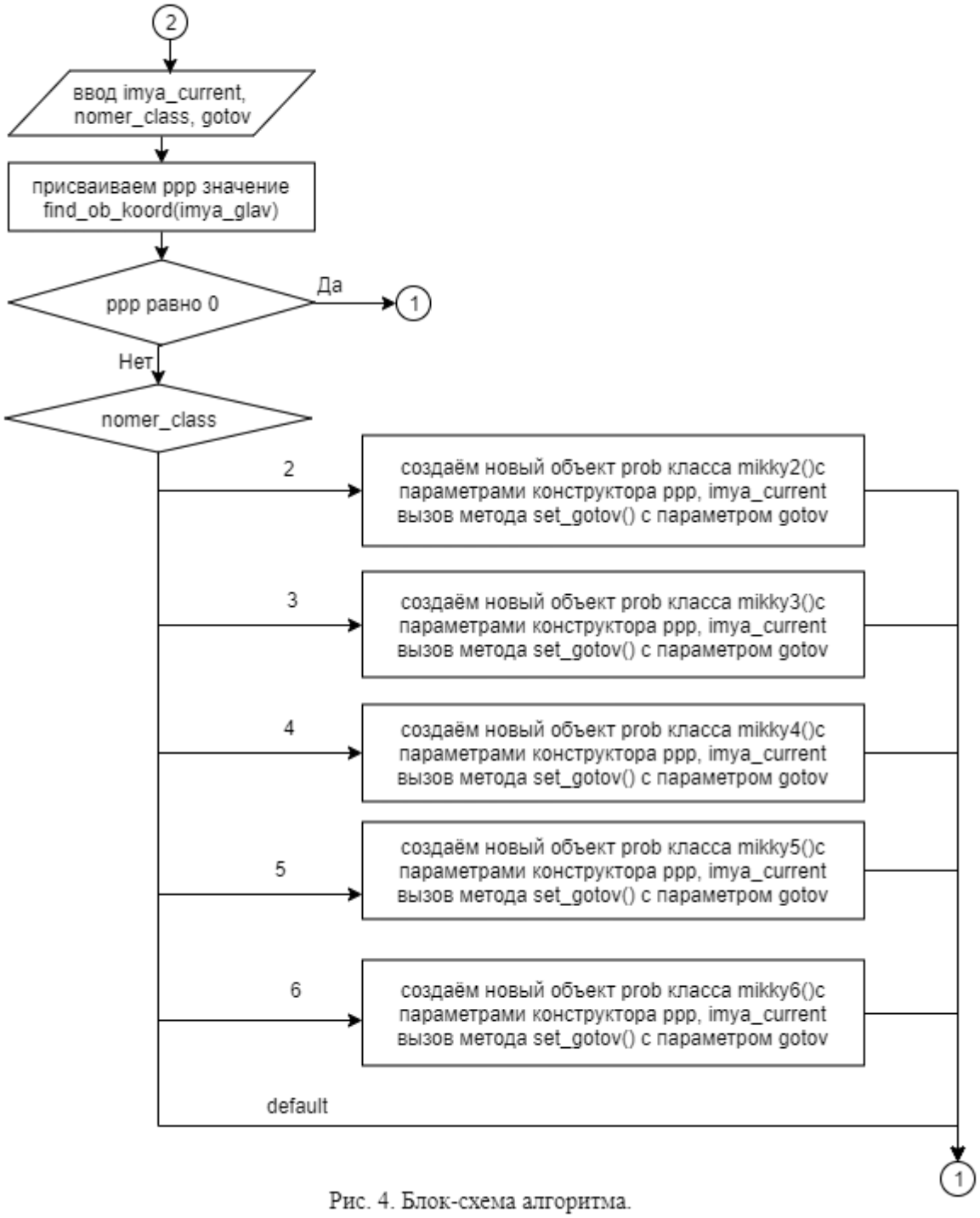

Рис. 4. Блок-схема алгоритма.

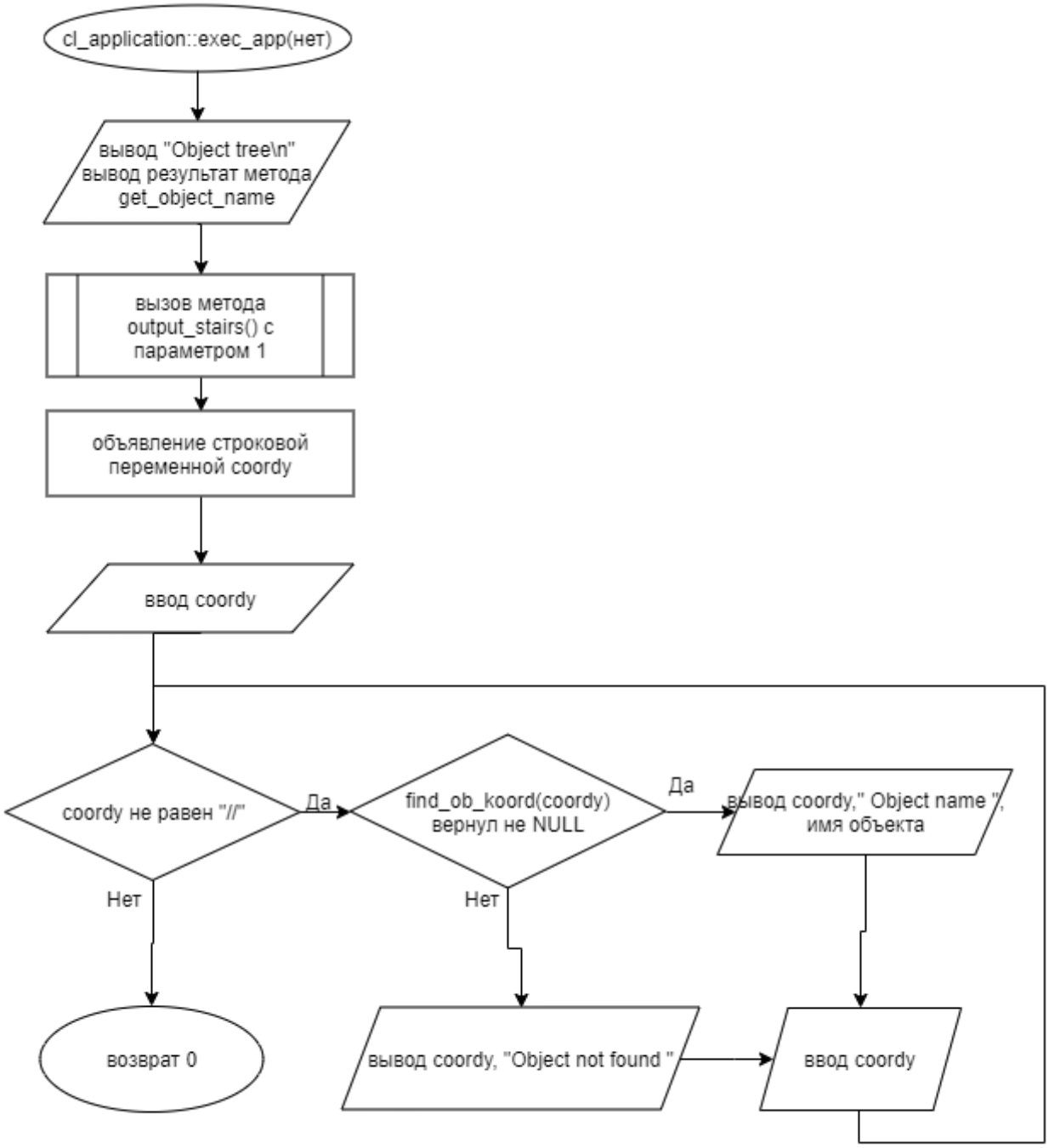

Рис. 5. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл cl_application.cpp

```
#include <iostream>
#include "cl_base.h"
#include "cl_application.h"
#include "mikky2.h"
#include "mikky3.h"
#include "mikky4.h"
#include "mikky5.h"
#include "mikky6.h"


#include <string>


using namespace std;

cl_application::cl_application(cl_base* p_parent):
        cl_base (p_parent)
{
}

void cl_application::bild_tree_objects()
{
        string imya_first, imya_glav, imya_current;
        int nomer_class, gotov;
        cin>>imya_first;
        set_object_name(imya_first);
        set_gotov(1);
        cl_base* ppp = this;
        cl_base* prob = nullptr;
        do
        {
                cin>>imya_glav;
                if( imya_glav=="endtree")
                        return;
                else
                cin>>imya_current>>nomer_class>>gotov;

                ppp = find_ob_koord(imya_glav);
                if (ppp)
                {
                        switch(nomer_class)
                        {
                                case 2:
                                        prob = new mikky2(ppp, imya_current);
                                        prob->set_gotov(gotov);
                                        break;
                                case 3:
                                        prob = new mikky3(ppp, imya_current);
```

```
                                        prob->set_gotov(gotov);
                                        break;
                                case 4:
                                        prob = new mikky4(ppp, imya_current);
                                        prob->set_gotov(gotov);
                                        break;
                                case 5:
                                        prob = new mikky5(ppp, imya_current);
                                        prob->set_gotov(gotov);
                                        break;
                                case 6:
                                        prob = new mikky6(ppp, imya_current);
                                        prob->set_gotov(gotov);
                                        break;
                        }
                }
        }while(true);

}

int cl_application::exec_app()
{
        cout<<"Object tree\n";
        cout<<get_object_name();
        output_stairs(1);

        string coordy;
        cin>>coordy;
        while(coordy!="//")
        {
                if (find_ob_koord(coordy))
                        cout<<endl<<coordy<<" Object name:
"<<(find_ob_koord(coordy))->get_object_name();
                else
                        cout<<endl<<coordy<<" Object not found";
                cin>>coordy;
        }
        return 0;
}
```

Файл cl_application.h

```
#ifndef HEADER1_H
#define HEADER1_H
#include <string>
#include <vector>
#include "cl_base.h"

using namespace std;
class cl_application: public cl_base
{
public:
        cl_application(cl_base*);
```

```
        void bild_tree_objects();
        int exec_app();
};

#endif
```

Файл cl_base.cpp

```
#include <iostream>
#include "cl_base.h"
#include "cl_application.h"
#include "mikky2.h"
#include "mikky3.h"
#include "mikky4.h"
#include "mikky5.h"
#include "mikky6.h"

#include <string>


using namespace std;

cl_base::cl_base(cl_base* p, string str )
{
        this->object_name = str;
        parent=p;
                if (parent)
                        parent->children.push_back(this);
}

cl_base* cl_base::find_ob(string name)
{
        if (get_object_name()==name) return this;
        cl_base* temp;
        if(children.empty()) return nullptr;
        child_iterator = children.begin();
        while (child_iterator != children.end())
        {
                if((*child_iterator)->get_object_name() ==name)
                        return (*child_iterator);
                else
                {
                        temp = (*child_iterator)->find_ob(name);
                        if (temp)
                                return temp;
                }
                child_iterator++;
        }
        return nullptr;
}

cl_base* cl_base::find_ob_koord(string name)
{
        if (name[1]=='/')
                return find_ob(name.erase(0,2));
```

```
        string temp;
        for (int i =1; i<name.size();i++)
        {
                if (name[i]=='/')
                {
                        break;
                }
                else
                        temp+=name[i];
        }
        if (this->get_object_name()!=temp)
                return nullptr;
        name.erase(0,temp.size()+1);
        if (name.empty())
                return this;
        temp="";
        for (int i = 1; i<name.size();i++)
        {
                if (name[i]=='/')
                {
                        break;
                }
                else
                        temp+=name[i];
        }
        this->child_iterator = this->children.begin();
        while (this->child_iterator != this->children.end())
        {
                if ((*this->child_iterator)->get_object_name() == temp)
                {
                        return (*this->child_iterator)->find_ob_koord(name);
                }
                this->child_iterator++;
        }
        return nullptr;
}


void cl_base::set_gotov(int gotov)
{
        this->gotov=gotov;
}

int cl_base::get_gotov()
{
        return this->gotov;
}


void cl_base::output_gotov()
{
        if (this->gotov >0)
                cout<<endl<<"The object "<<get_object_name()<<" is ready";
        else
                cout<<endl<<"The object "<<get_object_name()<<" is not ready";
        if(children.empty()) return;
        child_iterator=children.begin();
```

```
        while (child_iterator!=children.end())
        {
                (*child_iterator)->output_gotov();
                child_iterator++;
        }
}


void cl_base::set_object_name(string object_name)
{
        this->object_name = object_name;
}

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

void cl_base::set_parent(cl_base* new_parent)
{
        if (parent)
        {
                for(parent->child_iterator = parent->children.begin(); parent-
>child_iterator != children.end();parent->child_iterator++)
                {
                        if ((*parent->child_iterator) == this)
                        {
                                parent->children.erase(parent-
>child_iterator);
                                break;
                        }
                }
        }
                parent=new_parent;
                new_parent->children.push_back(this);
}

cl_base* cl_base::get_parent()
{
        return this->parent;
}

void cl_base::output()
{
        if (children.size()!=0)
        {
                cout<<endl<<object_name;
                for(int i=0; i<children.size(); i++)
                        cout<< "  "<<children[i]->object_name;
                for(int i=0; i<children.size(); i++)
                        children[i]->output();
        }
}

void cl_base::output_stairs(int level)
{
        string p(level*4, ' ');
        if(children.empty()) return;
        child_iterator=children.begin();
        while (child_iterator!=children.end())
```

```
        {
                cout<<endl<<p<<(*child_iterator)->get_object_name();
                (*child_iterator)->output_stairs(level+1);
                child_iterator++;
        }
}

cl_base::~cl_base()
{
        for (int i=0; i<children.size(); i++)
                delete children[i];
}
```

Файл cl_base.h

```
#ifndef HEADER_H
#define HEADER_H
#include <string>
#include <vector>

using namespace std;

class cl_base
{
public:

        cl_base(cl_base* parent, string str = "cl_base");
        void set_object_name(string name);
        string get_object_name();
        cl_base* get_parent();
        void set_parent(cl_base*);
        void output();

        cl_base* find_ob(string);
        cl_base* find_ob_koord(string);

        void output_gotov();
        void output_stairs(int level);
        void set_gotov(int);
        int get_gotov();

        ~cl_base();

private:
        int gotov;
        vector <cl_base*> children;
        vector <cl_base*>::iterator child_iterator;
        string object_name;
        cl_base* parent;
};

#endif
```

Файл cl_kids.cpp

```
#include <iostream>
#include "cl_base.h"
#include "cl_application.h"
#include "cl_kids.h"

#include <string>

using namespace std;

cl_kids::cl_kids(cl_base* p_parent, string str):
        cl_base (p_parent, str)
{
}
```

Файл cl_kids.h

```
#ifndef HEADER2_H
#define HEADER2_H
#include <string>
#include <vector>
#include "cl_base.h"

using namespace std;

class cl_kids:public cl_base
{
public:
        cl_kids(cl_base*,string str);
};
#endif
```

Файл main.cpp

```
#include <iostream>
#include <string>
#include "cl_base.h"
#include "cl_application.h"
#include "cl_kids.h"


using namespace std;

int main()
{
        cl_application ob_cl_application(nullptr);
        ob_cl_application.bild_tree_objects();
```

```
        return ob_cl_application.exec_app();

}
```

Файл mikky2.cpp

```
#include <iostream>
#include "cl_base.h"
#include "cl_application.h"
#include "mikky2.h"



#include <string>

using namespace std;

mikky2::mikky2(cl_base* p_parent, string str):
        cl_base (p_parent, str)
{
}
```

Файл mikky2.h

```
#ifndef HEADER21_H
#define HEADER21_H
#include <string>
#include <vector>
#include "cl_base.h"

using namespace std;

class mikky2: public cl_base
{
public:
        mikky2(cl_base*,string str);
};
#endif
```

Файл mikky3.cpp

```
#include <iostream>
#include "cl_base.h"
#include "cl_application.h"
#include "mikky3.h"
```

```
#include <string>

using namespace std;

mikky3::mikky3(cl_base* p_parent, string str):
        cl_base (p_parent, str)
{
}
```

Файл mikky3.h

```
#ifndef HEADER31_H
#define HEADER31_H
#include <string>
#include <vector>
#include "cl_base.h"


using namespace std;

class mikky3:public cl_base
{
public:
        mikky3(cl_base*,string str);
};
#endif
```

Файл mikky4.cpp

```
#include <iostream>
#include "cl_base.h"
#include "cl_application.h"
#include "mikky4.h"

#include <string>

using namespace std;

mikky4::mikky4(cl_base* p_parent, string str):
        cl_base (p_parent, str)
{
}
```

Файл mikky4.h

```
#ifndef HEADER41_H
```

```
#define HEADER41_H
#include <string>
#include <vector>
#include "cl_base.h"


using namespace std;

class mikky4:public cl_base
{
public:
        mikky4(cl_base*,string str);
};
#endif
```

Файл mikky5.cpp

```
#include <iostream>
#include "cl_base.h"
#include "cl_application.h"
#include "mikky5.h"

#include <string>

using namespace std;

mikky5::mikky5(cl_base* p_parent, string str):
        cl_base (p_parent, str)
{
}
```

Файл mikky5.h

```
#ifndef HEADER51_H
#define HEADER51_H
#include <string>
#include <vector>
#include "cl_base.h"


using namespace std;

class mikky5:public cl_base
{
public:
        mikky5(cl_base*,string str);
};
#endif
```

Файл mikky6.cpp

```
#include <iostream>
#include "cl_base.h"
#include "cl_application.h"
#include "mikky6.h"

#include <string>

using namespace std;

mikky6::mikky6(cl_base* p_parent, string str):
        cl_base (p_parent, str)
{
}
```

Файл mikky6.h

```
#ifndef HEADER61_H
#define HEADER61_H
#include <string>
#include <vector>
#include "cl_base.h"


using namespace std;

class mikky6:public cl_base
{
public:
        mikky6(cl_base*,string str);
};
#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
r /r y 3 1 endtree //y //	Object tree r y //y Object name: y	Object tree r y //y Object name: y
r /r y 2 1 /r/y y 2 1 /r/y/y u 2 -1 /r i 3 1 endtree /r /r/i /r/o //	Object tree r y y u i /r Object name: r /r/i Object name: i /r/o Object not found	Object tree r y y u i /r Object name: r /r/i Object name: i /r/o Object not found
r /r o_8 2 1 endtree //o_5 //	Object tree r o_8 //o_5 Object not found	Object tree r o_8 //o_5 Object not found
root endtree /root //r //	Object tree root /root Object name: root //r Object not found	Object tree root /root Object name: root //r Object not found

ЗАКЛЮЧЕНИЕ

В результате выполнения работы я приобрел навыки написания программ с помощью ООП подхода на примере языка программирования C++. Объектно-ориентированный подход является наиболее распространёной методикой разработки на данный момент. Навык написания программ в этой парадигме является основополагающим для любого современного программиста. На конкретных учебных задачах я научился разрабатывать базовый класс для объектов, определять общий функционал для используемых в рамках приложения объектов, разрабатывать операции добавления, удаления, изменения позиции объекта в рамках иерархического дерева, построению дерева иерархии объектов, переключению состояния объектов и определения их готовности к работе, выводить на печать дерево иерархии объектов, искать указатель на объект по координате на дереве иерархии объектов, или по имени, в случае уникальности наименования объектов.

Также я получил навыки проектирования, написания, отладки и модификации программ в ООП стиле. Были изучены стандартные конструкции и основные библиотеки языка C++, и усвоен материал, изложенный в учебных пособиях по данной теме.

В данной работе мной был реализован метод поиска объекта по его координате в дереве с целью получения его адреса. Данные работы позволили мне закрепить знания, полученные во время лекций, значительно улучшили навык написания программ, вдохновили меня на продолжение самостоятельного изучения ООП.

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).