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_2 Определение указателя на объект по его координате »

С тудент группы	ИКБО-33-21	Евсеев Г.Е.
Руководитель практики	Старший преподаватель	Грач Е.П.
Работа представлена	«___»_____2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Постановка задачи

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 согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задается в следующем виде:

/ - корневой объект;
//«имя объекта» - поиск объекта по уникальному имени от корневого (для однозначности уникальность требуется в рамках дерева);
. - текущий объект;
«имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;
/«имя объекта 1»[/«имя объекта 2»] . . . - абсолютная координата от корневого объекта.

Примеры координат:

```
/
//ob_3
.
ob_2/ob_3
ob_2
/ob_1/ob_2/ob_3
```

Если координата пустая строка или объект не найден, то вернуть нулевой указатель.

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.
При построении дерева уникальность наименования относительно множества непосредственно подчиненных объектов для любого головного объекта соблюдены.
Добавить проверку допустимости исходной сборки. Собрать дерево невозможно, если по заданной координате головной объект не найден (например, ошибка в наименовании или еще не расположен на дереве объектов).

Система отрабатывает следующие команды:
SET «координата» – устанавливает текущий объект;
FIND «координата»– находит объект относительно текущего;
END – завершает функционирование системы (выполнение программы).
Изначально, корневой объект для системы является текущим.
При вводе данных в названии команд ошибок нет. Условия уникальности имен объектов для однозначной отработки соответствующих команд соблюдены.

Описание входных данных

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

После ввода состава дерева иерархии построчно вводятся команды:

SET «координата» - установить текущий объект;

FIND «координата» - найти объект относительно текущего;

END – завершить функционирование системы (выполнение программы).

Команды SET и FIND вводятся произвольное число раз. Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов.

```
root
/                object_1                3
/                object_2                2
/object_2             object_4              3
/object_2             object_5              4
/                object_3                3
/object_2             object_3              6
/object_1             object_7              5
/object_2/object_4          object_7           3
endtree
FIND                  object_2/object_4
SET                      /object_2
```

```
FIND                    //object_5
FIND                    /object_15
FIND                    .
FIND                  object_4/object_7
END
```

Описание выходных данных

Первая строка:

`Object tree`

Со второй строки вывести иерархию построенного дерева как в курсовой работе версия №2.

При ошибке определения головного объекта, прекратить сборку, вывести иерархию уже построенного фрагмента дерева, со следующей строки сообщение:

`The head object «координата головного объекта» is not found`

и прекратить работу программы.

Если дерево построено, то далее построчно:

для команд SET если объект найден, то вывести:

`Object is set: «имя объекта»`

в противном случае:

`Object is not found: «имя текущего объекта»«искомая координата объекта»`

для команд FIND вывести:

«искомая координата объекта» Object name: «наименование объекта»

Если объект не найден, то: «искомая координата объекта» Object is not found

Пример вывода иерархии дерева объектов.

Object tree

root

object_1

object_7

object_2

object_4

object_7

object_5

object_3

object_3

object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7

Метод решения

Для решения задачи используются:

- контейнер для хранения пар данных pair;
- Класс cl_base:
  - Методы:
    - Метод find_object_by_coord:
      - Функционал - возвращает указатель на объект по описанному пути, если нет - пустой указатель;
    - Метод get_children:
      - Функционал - возвращает вектор указателей на детей данного объекта;

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: get_children

Функционал: возвращает вектор указателей на детей данного объекта

Параметры: отсутствуют

Возвращаемое значение: вектор указателей на cl_base

Алгоритм метода представлен в таблице 1.

Таблица 1. Алгоритм метода get_children класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		возвращает поле children данного объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: find_object_by_coord

Функционал: возвращает указатель на объект по описанному пути, если нет - пустой указатель

Параметры: указатель на cl_base current_object, root_object, строка coord

Возвращаемое значение: указатель на cl_base

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода find_object_by_coord класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1	coord = ""	возвращает пустой указатель	∅	
	coord = "."	возвращает текущий объект	∅	
	coord = "/"	возвращает корневой объект	∅	
	coord = "//"	объявление строки name	2	
			4	
2	i < длины coord	добавление name строки coord[i]	2	
			3	
3		возвращает результат вызова метода find_object_by_name от root_object	∅	
4		объявление указателя cl_base new_object	5	
5		объявление строки name	6	
6		объявление целого i	7	
7	coord[i] = '/'	присваивание now_object = root_object, i = 1	8	
		присваивание now_object = current_object, i = 0	8	
8	i < длина coord		9	
		возвращает now_object	∅	
9	coord[i] == '/' или i == длина coord - 1		10	
		добавление к name	9	

		строки coord[i]		
10	i = длина coord - 1	добавление к name строки coord[i]	11	
			11	
11	name == "."	переход к следующему шагу цикла	8	
			12	
12		объявление логической found = false	13	
13	перебор детей now_object с помощью pointer		14	
			16	
14	имя pointer = name	присваивание now_object = pointer	15	
			13	
15		присваивание name = "", found = true	13	
16	фаунд = ложь	возвращает пустой указатель	∅	
			8	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

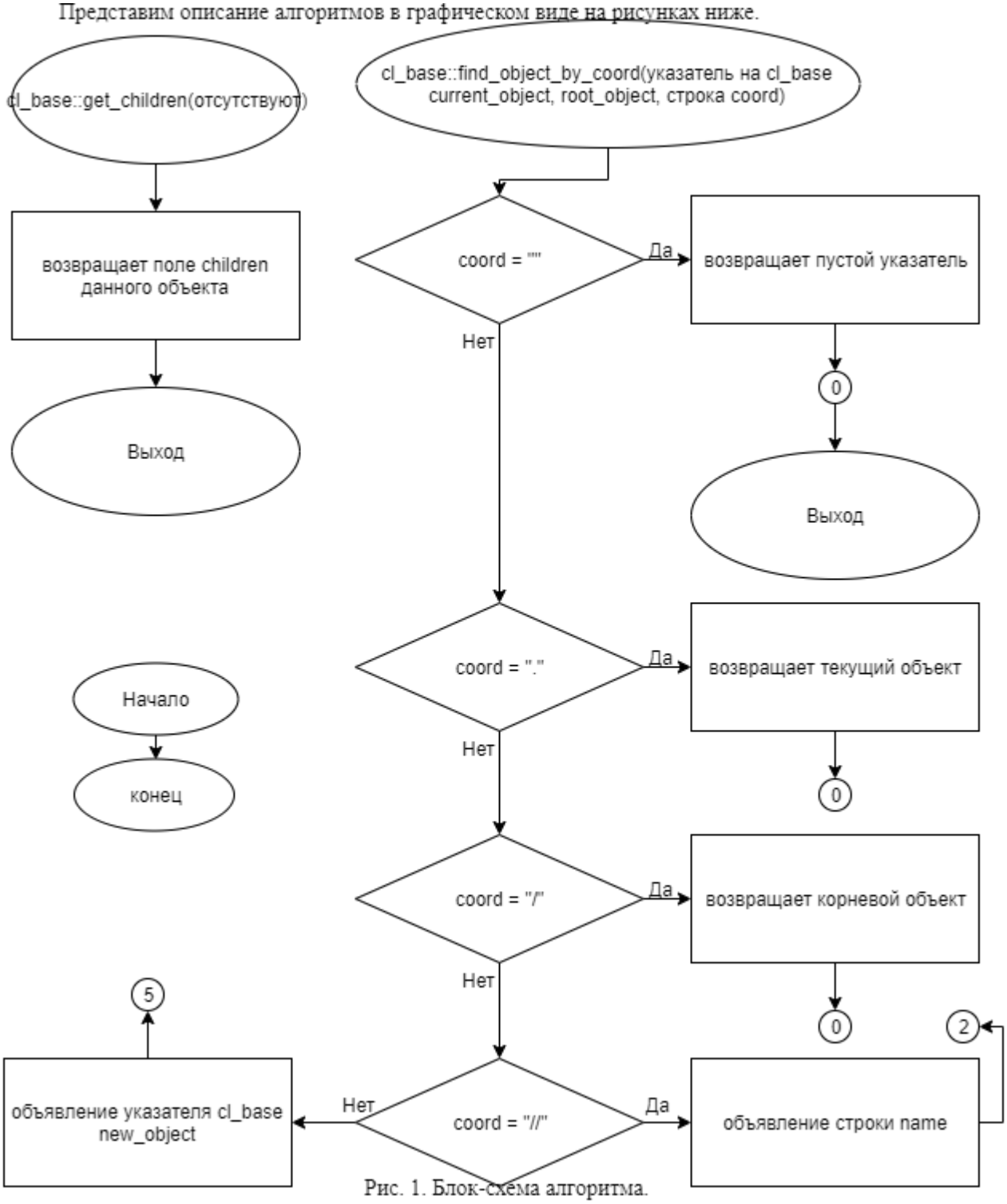

Рис. 1. Блок-схема алгоритма.

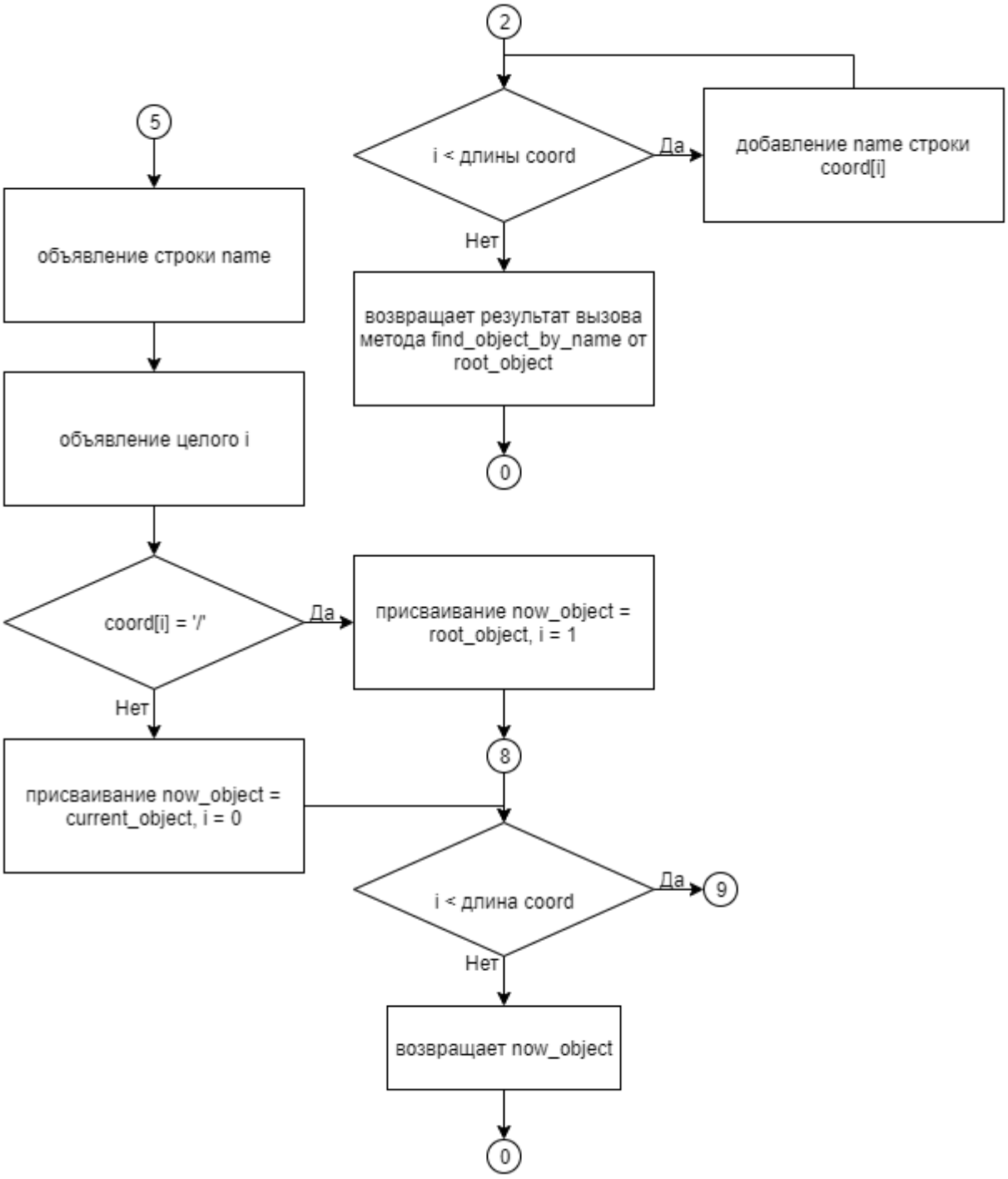

Рис. 2. Блок-схема алгоритма.

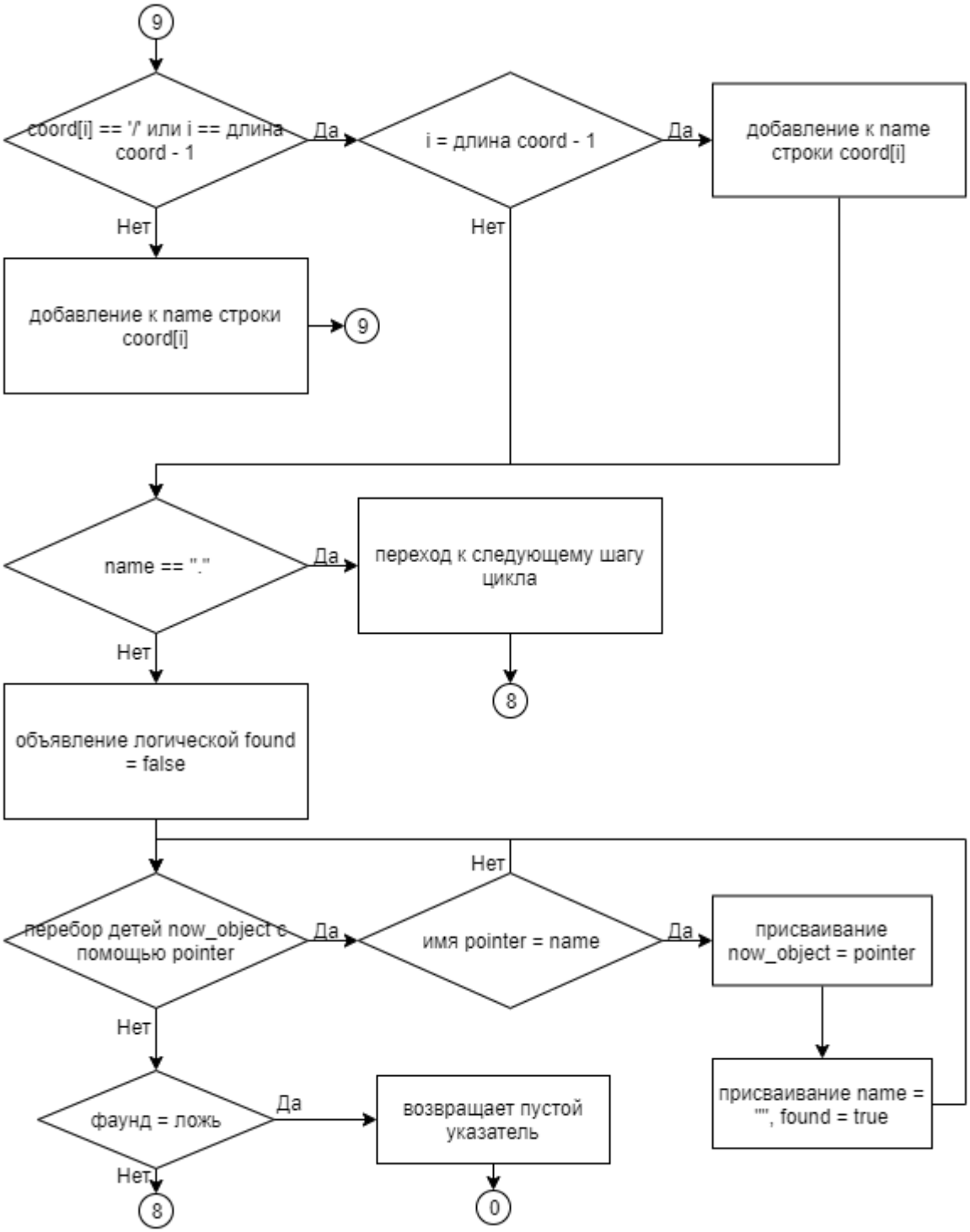

Рис. 3. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл cl_1.h

```
#ifndef __CL_1_H__
#define __CL_1_H__
#include "cl_base.h"
class cl_1 : public cl_base
{
public:
        cl_1(string name, cl_base* parent) : cl_base(name, parent) {};
};
#endif
```

Файл cl_2.h

```
#ifndef __CL_2_H__
#define __CL_2_H__
#include "cl_base.h"
class cl_2: public cl_base
{
public:
        cl_2(string name, cl_base* parent): cl_base(name, parent) {};
};
#endif
```

Файл cl_3.h

```
#ifndef __CL_3_H__
#define __CL_3_H__
#include "cl_base.h"
class cl_3: public cl_base
{
public:
        cl_3(string name, cl_base* parent): cl_base(name, parent) {};
};
#endif
```

Файл cl_4.h

```
#ifndef __CL_4_H__
#define __CL_4_H__
#include "cl_base.h"
class cl_4: public cl_base
{
public:
        cl_4(string name, cl_base* parent): cl_base(name, parent) {};
};
#endif
```

Файл cl_5.h

```
#ifndef __CL_5_H__
#define __CL_5_H__
#include "cl_base.h"
class cl_5: public cl_base
{
public:
        cl_5(string name, cl_base* parent): cl_base(name, parent) {};
};
#endif
```

Файл cl_application.cpp

```
#include "cl_application.h"
pair<bool, string> cl_application::build_tree_objects()
{
        string parent_name, child_name;
        cl_base* parent, * child;

        cin >> parent_name;
        set_name(parent_name);

        int class_number;

        while (true)
        {
                cin >> parent_name;

                if (parent_name == "endtree")
                        return make_pair(true, "");

                cin >> child_name >> class_number;
                parent = find_object_by_coord(this, this, parent_name);
                        if (parent == nullptr)
                                return make_pair(false, parent_name);
                switch (class_number)
                {
                case(2):
```

```
                        child = new cl_1(child_name, parent);
                        break;
                case(3):
                        child = new cl_2(child_name, parent);
                        break;
                case(4):
                        child = new cl_3(child_name, parent);
                        break;
                case(5):
                        child = new cl_4(child_name, parent);
                        break;
                case(6):
                        child = new cl_5(child_name, parent);
                        break;
                }
        }
}
int cl_application::exec_app(bool is_okay, string not_okay_name)
{
        if (is_okay)
        {
                cout << "Object tree";
                print_new("");
                string operation, coord;
                cl_base* current_object = this;
                while (true)
                {
                        cin >> operation;
                        if (operation == "END")
                                break;
                        cin >> coord;
                        if (operation == "SET")
                        {
                                cl_base* result =
find_object_by_coord(current_object, this, coord);
                                if (result == nullptr)
                                        cout << endl << "Object is not found:
" << current_object->get_name() << ' ' << coord;
                                else
                                {
                                        cout << endl << "Object is set: " <<
result->get_name();
                                        current_object = result;
                                }
                        }
                        else if (operation == "FIND")
                        {
                                cl_base* result
=find_object_by_coord(current_object, this, coord);
                                cout << endl << coord << "     ";
                                if (result == nullptr)
                                        cout << "Object is not found";
                                else
                                        cout << "Object name: " << result ->
get_name();
                        }
                }
        }
        else
        {
```

```
                cout << "Object tree";
                print_new("");
                cout << endl << "The head object " << not_okay_name << " is
not found";
        }

        return 0;
}
```

Файл cl_application.h

```
#ifndef __CL_APPLICATION_H__
#define __CL_APPLICATION_H__
#include "cl_base.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
class cl_application : public cl_base
{
public:
        cl_application(string name = "") : cl_base(name) {};
        pair<bool, string> build_tree_objects();
        int exec_app(bool, string);
};
#endif
```

Файл cl_base.cpp

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

```
{
        this->name = name;
        return;
}
void cl_base::set_parent(cl_base* new_parent)
{
        if (parent)
                for (int i = 0; i < parent->children.size(); i++)
                        if (parent->children[i] == this)
                                parent->children.erase(parent ->
children.begin() + i);
        parent = new_parent;
        if (new_parent)
                new_parent->children.push_back(this);
        return;
}
cl_base* cl_base::get_parent()
{
        return this->parent;
}
void cl_base::print()
{
        if (children.size())
        {
                cout << endl << get_name();
                for (int i = 0; i < children.size(); i++)
                        cout << "    " << children[i]->get_name();
                for (int i = 0; i < children.size(); i++)
                        children[i]->print();
        }
        return;
}
void cl_base::print_new(string depth)
{
        cout << endl << depth << get_name();
        for (auto child : children)
                child->print_new(depth + "    ");
}
void cl_base::print_new_state(string depth)
{
        cout << endl << depth << get_name();
        if (!state)
                cout << " is not ready";
        else
                cout << " is ready";
        for (cl_base* child : children)
                child->print_new_state(depth + "    ");
}
cl_base* cl_base::find_object_by_name(string name)
{
        if (this->name == name)
                return this;
        else
        {
                for (cl_base* child : children)
                {
                        cl_base* result = child->find_object_by_name(name);
                        if (result)
                                return result;
                }
```

```
                return nullptr;
        }
}
int cl_base::get_state()
{
        return this->state;
}
bool cl_base::set_state(int state)
{
        if (!state)
                if ((parent && parent->get_state() == 0) || get_state() == 0)
                        return true;
                else
                {
                        for (cl_base* child : children)
                                child->set_state(0);
                        this->state = 0;
                        return true;
                }
        else
                if (parent && parent->get_state() == 0)
                        return false;
                else
                {
                        this->state = state;
                        return true;
                }
}
vector<cl_base*> cl_base::get_children()
{
        return this->children;
}
cl_base* cl_base::find_object_by_coord(cl_base* current_object,
cl_base*root_object, string coord)
{
        if (coord == "")
                return nullptr;

        if (coord == ".")
                return current_object;

        if (coord == "/")
                return root_object;

        if (coord[0] == '/' && coord[1] == '/')
        {
                string name = "";
                for (int i = 2; i < coord.length(); ++i)
                        name += coord[i];

                return root_object->find_object_by_name(name);
        }
        cl_base* now_object;
        string name = "";
        int i;

        if (coord[0] == '/')
```

```
        {
                now_object = root_object;
                i = 1;
        }
        else
        {
                now_object = current_object;
                i = 0;
        }

        for (; i < coord.length(); ++i)
        {
                if (coord[i] == '/' || i == coord.length() - 1)
                {
                        if (i == coord.length() - 1)
                                name += coord[i];

                        if (name == ".")
                                continue;

                        bool found = false;

                        for (cl_base* pointer : now_object->get_children())
                                if (pointer->get_name() == name)
                                {
                                        now_object = pointer;
                                        name = "";
                                        found = true;
                                }

                        if (!found)
                                return nullptr;
                }
                else
                        name += coord[i];

        }
        return now_object;
}
```

Файл cl_base.h

```
#ifndef __CL_BASE_H__
#define __CL_BASE_H__
#include <iostream>
#include <vector>
#include <string>
using namespace std;
class cl_base
{
        int state = 0;
        string name;
        cl_base* parent;
        vector<cl_base*> children;
public:
        cl_base(string name = "cl_base", cl_base* parent = nullptr);
```

```
        ~cl_base();
        string get_name();
        void set_name(string& name);
        cl_base* get_parent();
        void set_parent(cl_base* new_parent);
        void print();
        void print_new(string);
        void print_new_state(string);
        virtual cl_base* find_object_by_name(string name);
        bool set_state(int state);
        int get_state();
        cl_base* find_object_by_coord(cl_base*, cl_base*, string);
        vector<cl_base*> get_children();
};
#endif
```

Файл main.cpp

```
#include "cl_application.h"
int main()
{
        cl_application application;
        pair<bool, string> result_of_build = application.build_tree_objects();
        return
application.exec_app(result_of_build.first,result_of_build.second);
}
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 SET /object_2 FIND //object_5 FIND /object_15 FIND . FIND object_4/object_7 END	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7
root / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 SET /object_2 FIND //object_5 FIND /object_15 FIND . FIND object_4/object_7 END	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7
root / object_1 3 / object_2	Object tree root object_1	Object tree root object_1

2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 SET /object_2 FIND //object_5 FIND /object_15 FIND . FIND object_4/object_7 END	object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7	object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7

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).