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 строиться 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 в контрольной работе № 1.
После ввода состава дерева иерархии построчно вводится 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

В базовый класс Base добавлен новый метод pNext() - определить указатель на объект по его координате

Изменен метод execute() - запускающий приложение

Описание алгоритма

Класс объекта: Application

Модификатор доступа: public

Метод: execute

Функционал: Запуск основного алгоритма приложения

Параметры: нет

Возвращаемое значение: Целое - индикатор корректности завершения метода

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Object tree" и	2	
2		Перевод на новую строку	3	
3		Вызов метода printSpace()	4	Параметры метода: 0
4		Объявление строковых переменных s1 и s2	5	
5		Ввод строковой переменной s1	6	
6	s1 не равно "//"	Перевод на новую строку	7	
			∅	
7	s1 не равно "//"	Вызов метода getByName с аргументом s1	8	Оператор цикла - while
			∅	
8		Присваивание результата работы getByName переменной Base * b	9	
9	b не нулевой указатель	Вывести "Object name: " и перевод на новую строку	10	
		Вывести Object not found и перевод на новую строку	10	
10	s2 не равно "//"	Перевод на новую строку	11	
			∅	
11	s1 = s2	Присвоение значение строковой переменной s1	7	

Класс объекта: Base

Модификатор доступа: public

Метод: pNext

Функционал: Функция для поиска вершины

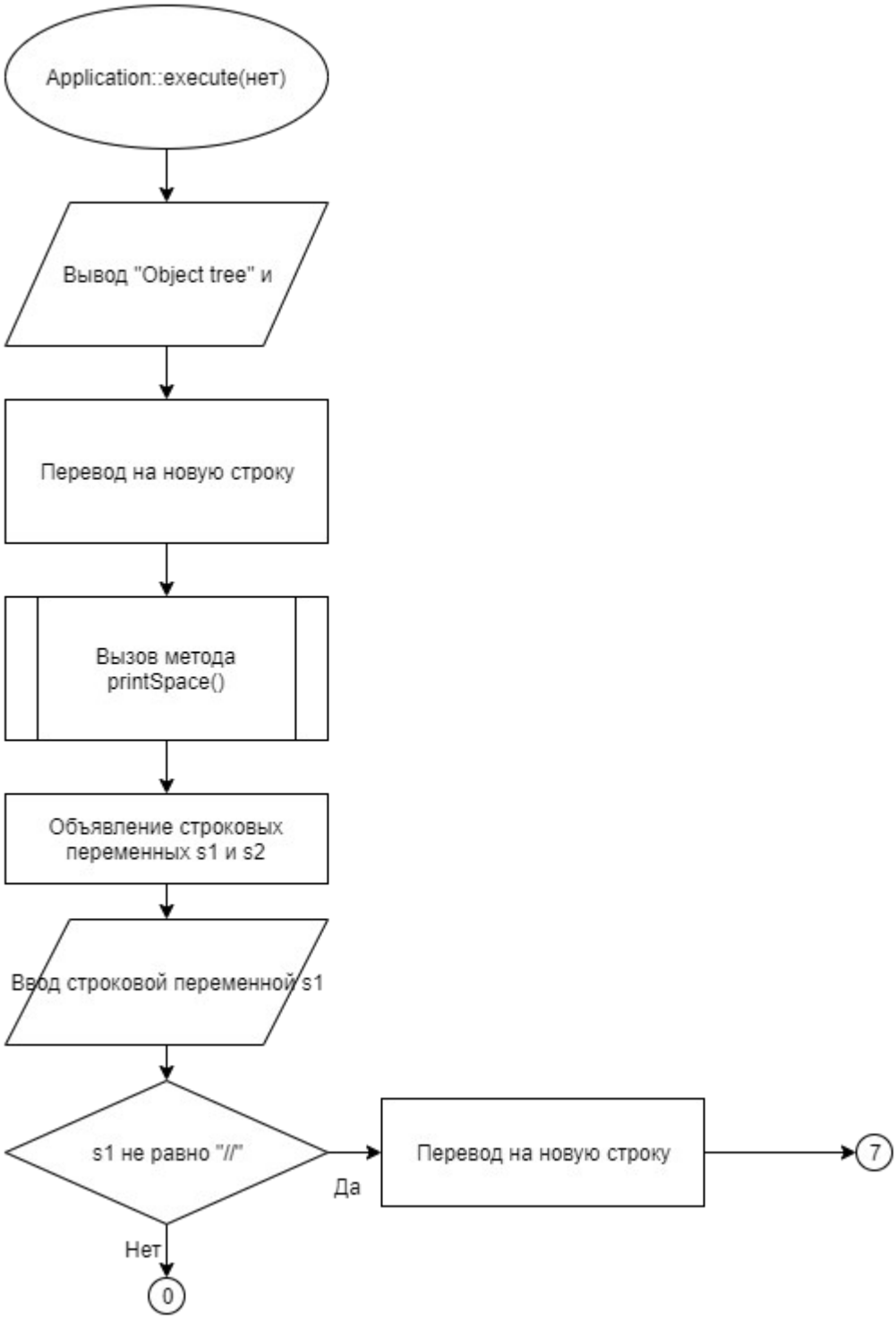
Параметры: coordinates - путь к искомой вершины

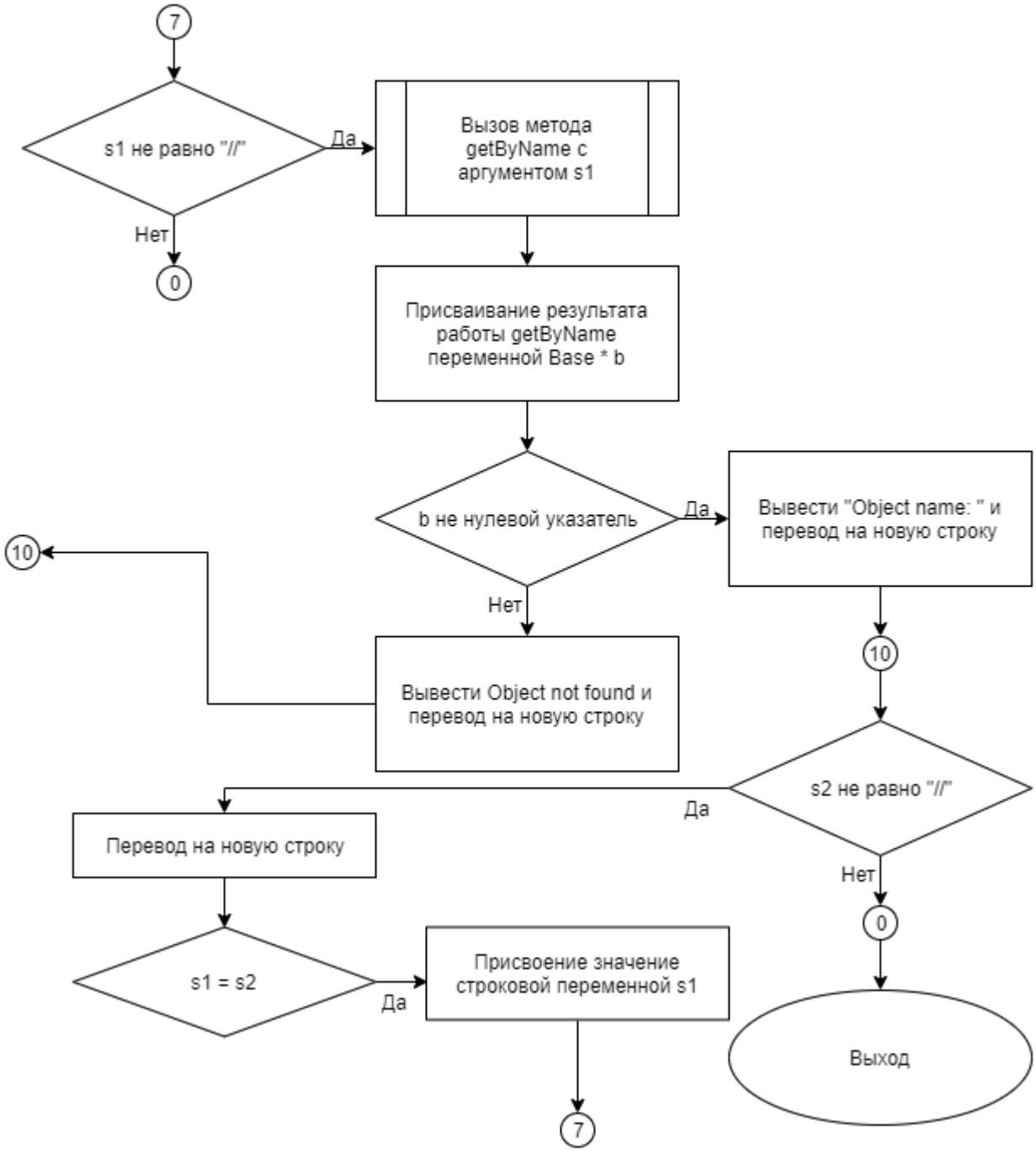
Возвращаемое значение: Указатель на нужную вершину или нулевой указатель

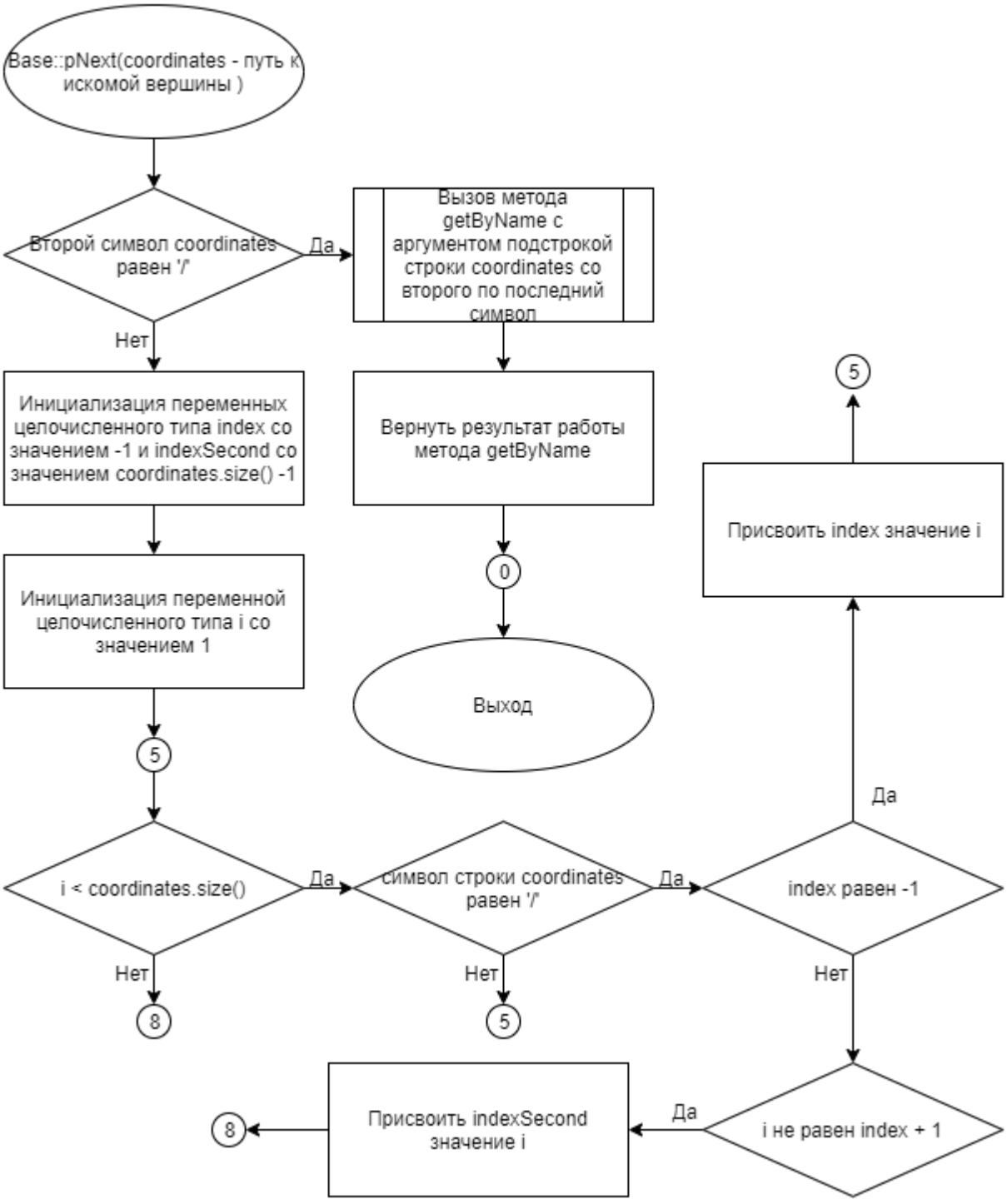
№	Предикат	Действия	№ перехода	Комментарий
1	Второй символ coordinates равен '/'	Вызов метода getByName с аргументом подстрокой строки coordinates со второго по последний символ	2	
			3	
2		Вернуть результат работы метода getByName	∅	
3		Инициализация переменных целочисленного типа index со значением -1 и indexSecond со значением coordinates.size() -1	4	
4		Инициализация переменной целочисленного типа i со значением 1	5	
5	i < coordinates.size()		6	
			8	
6	символ строки coordinates равен '/'		7	
			5	
7	index равен -1	Присвоить index значение i	5	
	i не равен index + 1	Присвоить indexSecond значение i	8	
8	index равен -1		∅	
			9	
9		Иициализация переменой строки s2 со значением подстроки coordinates с index + 1 символа по indexSecond	10	
10		Присвоить целочисленной переменной 0	11	
11	i меньше кол-во наследников		12	
		Вернуть нулевой указатель	∅	

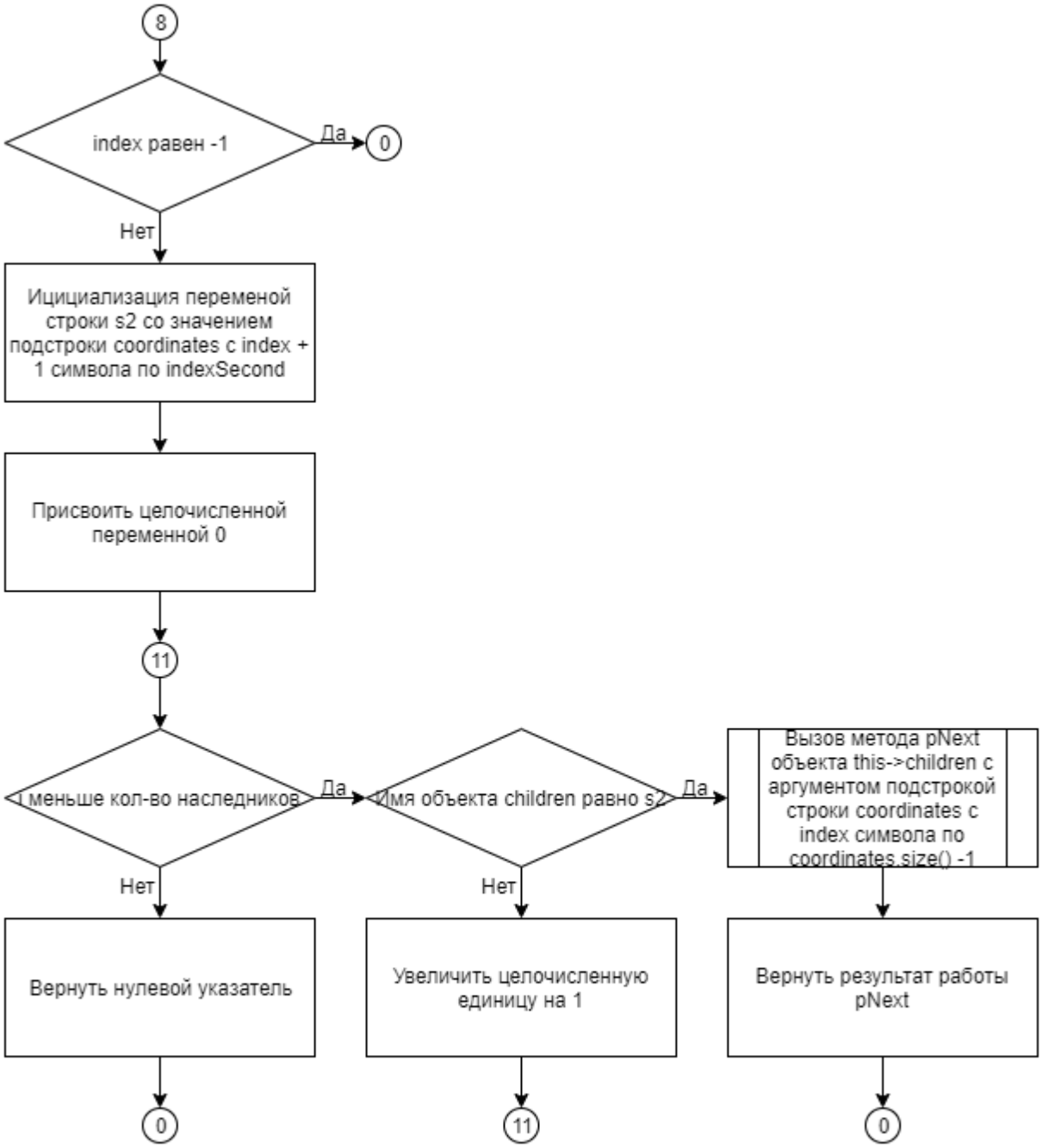
12	Имя объекта children равно s2	Вызов метода pNext объекта this->children с аргументом подстрокой строки coordinates с index символа по coordinates.size() -1	13	
		Увеличить целочисленную единицу на 1	11	
13		Вернуть результат работы pNext	∅	

Блок-схема алгоритма


Application::execute(нет)
Вывод "Object tree" и
Перевод на новую строку
Вызов метода printSpace()
Объявление строковых переменных s1 и s2
Ввод строковой переменной s1
s1 не равно "//"
Да
Перевод на новую строку
7
Нет
0


7
s1 не равно "//"
Да
Вызов метода getByName с аргументом s1
Нет
0
Присваивание результата работы getByName переменной Base * b
b не нулевой указатель
Да
Вывести "Object name: " и перевод на новую строку
10
Нет
10
Вывести Object not found и перевод на новую строку
s2 не равно "//"
Да
Перевод на новую строку
Нет
0
s1 = s2
Да
Присвоение значение строковой переменной s1
Выход
7


Base::pNext(coordinates - путь к искомой вершины)
Второй символ coordinates равен '/'
Да
Вызов метода getByName с аргументом подстрокой строки coordinates со второго по последний символ
Нет
Инициализация переменных целочисленного типа index со значением -1 и indexSecond со значением coordinates.size() -1
Вернуть результат работы метода getByName
Инициализация переменной целочисленного типа i со значением 1
0
Выход
5
5
Присвоить index значение i
i < coordinates.size()
Да
символ строки coordinates равен '/'
Да
index равен -1
Да
Нет
Нет
Нет
8
5
Присвоить indexSecond значение i
Да
i не равен index + 1
8



Код программы

Файл application.cpp

```
#include "application.h"
#include "base.h"
#include "Derived.h"
#include "DerivedSecond.h" // new
#include "DerivedThird.h" // new
#include "DerivedFourth.h"
#include "DerivedFifth.h"
#include <iostream>
#include <string>
using namespace std;
Application:: Application(Base* parent):
        Base(parent,"",1) {}; // new
int Application::execute() { //
        //cout << this -> getName();
        //cout << this -> printNames();
        cout <<"Object tree\n"; //new
        this ->printSpace(0); //new
        string s1, s2;
        cin >> s1;
        if(s1 != "//")
                cout << "\n";
        while(s1 != "//") {
                Base * b = this -> pNext(s1);
                        cin >> s2;
                                cout << s1;
        if(b != NULL) {
                 cout << " Object name: " << b -> getName();
        } else {                        cout << " Object not found";

        }

        if(s2 != "//") {
                cout << "\n";
        }
        s1 = s2;
        }
        return 0;
}
void Application:: buildTree() {
        int status;
        string child_name, parent_name;
        int c = 0;
        cin >> parent_name;
        setName(parent_name);
        while(true) {
                cin >> parent_name;
                if(parent_name == "endtree") {
                        break;
                }
                cin >> child_name >> c >> status;

                Base* b = this -> pNext(parent_name); // new
                if(c == 2)
                                b -> add_child(new
```

```
Derived(b,child_name,status));
                else if(c == 3)          b -> add_child(new DerivedSecond(b,

child_name,status));
                else if(c == 4)          b -> add_child(new

child_name,status));
                else if(c == 5)          DerivedThird(b, b ->

child_name,status));
                else if(c == 6)          add_child(new

child_name,status));
        }                                DerivedFourth(b, b ->


                                         add_child(new


                                         DerivedFifth(b,


}
```

Файл application.h

```
#ifndef Application_h
#define Application_h
#include "base.h"
#include <iostream>

using namespace std;
class Application : public Base {
        public:
                                                    A
                                                    p
                                                    p
                                                    l
                                                    i
                                                    c
                                                    a
                                                    t
                                                    i
                                                    o
                                                    n
                                                    (
                                                    B
                                                    a
                                                    s
                                                    e
                                                    *
                                                    p
                                                    a
                                                    r
                                                    e
};
#endif
```

```
n                                                    e
t                                                    c
)                                                    u
;                                                    t
i                                                    e
n                                                    (
t                                                    )
e                                                    ;
x                                                    void buildTree();
```

Файл base.cpp

```
#include "base.h"
#include <iostream>
#include <vector>
#include <string>
using namespace std;

void Base:: setName(string new_name) {
        this -> name = new_name;
}
string Base:: getName() {
        return name;
}
void Base::setParent(Base* parent) {
```

```
        this -> parent = parent;
}
Base* Base:: getParent() {
        return parent;
}
Base::Base(Base* parent, string name, int status) // new
{
        setName(name);
        setParent(parent);
        this -> status = status;
}
void Base::printNames() {
        if(!children.size())
                return;
        cout << name;
        for(int i = 0; i < children.size(); ++i) {
                cout << " ";
                cout << children[i] -> getName();
        }
        for(int i = 0; i < children.size(); ++i) {
                if(children[i] -> children.size()) {
                        cout << endl;
                }
                children[i] -> printNames();
        }
}
void Base::tree() { // 3_1
        cout << "The object " << name;
        if(status > 0) {
                cout << " is ready";
        } else {                        cout << " is not ready";

        }

         if(children.size()) {
                 cout << endl;
        }
        // cout
        for(int i = 0; i < children.size(); ++i) {
                children[i] -> tree();
                if(i != children.size() -1)
                        cout << endl;
        }
}
void Base::printSpace(int space = 0) { // 3_2
        int spaceFour = space * 4;
        for(int i = 0; i < spaceFour; ++i)
                cout << " ";
        cout << this -> name;
        if(!children.size())
                return;
        cout << endl;
        for(int i = 0; i < children.size(); ++i) { children[i]
                -> printSpace(space + 1);
                        if(i!=children.size()-1) {
                                cout << endl;
                }
```

```
        }
        return;
}

void Base::add_child(Base *h) {
        children.push_back(h);
}
Base * Base::pNext(string coordinates) {
        if(coordinates[1] == '/') {
                return this -> getByName(coordinates.substr(2,
coordinates.size() - 2));
        }
        int index = -1;
        int indexSecond = coordinates.size(); for(int
        i = 1; i < coordinates.size(); ++i) {
                if(coordinates[i] == '/') {
                        if(index == -1)
                                index = i;
                                 else {  indexSe
                                         cond =
                                         i;
                                }            break;
                    }
             }

        if(index == -1)
                return this;
        string s2 = coordinates.substr(index + 1, indexSecond - index - 1);
        for(int i = 0; i <this->children.size(); ++i) {
                if(this -> children[i] -> getName() == s2) {
                        return this->children[i] ->
pNext(coordinates.substr(index, coordinates.size() - index));
                }
        }
        return nullptr;
}
Base* Base::getByName(string name) {
        if(this -> name == name) {
                return this;
        }
        for(int i = 0; i < children.size(); ++i) {
                Base* b = children[i] -> getByName(name);
                if(b!= NULL)
                        return b;
        }
}
```

Файл base.h

```
#ifndef Base_h
#define Base_h
#include <iostream>
#include <string>
#include <vector>
using namespace std;
```

```
class Base {
        protected:
                vector <Base*> children; // просто лекции грача, этот коммент
не удаляется
private:
        string name = "base"; //наименование
        Base* parent = 0; // ссылка на головной
        int status; // new special for 3_1
        public:
        Base(Base* parent, string name, int status); // new var
                void setName(string new_name);
                string getName();
                void setParent(Base* new_parent);
                Base* getParent();
                void printNames();
                void add_child(Base* h);
                Base * getByName(string name);
// labs                                         void tree(); // 3_1
                                                void printSpace(int space);//3_2
                                                Base * pNext(string coordinates); //
                                                3_3

};
#endif
```

Файл Derived.cpp

```
#include "Derived.h"
#include <iostream>
#include "base.h"
        Derived::Derived(Base* parent, string name, int status) : Base(parent,
name, status) {};
```

Файл DerivedFifth.cpp

```
#include "DerivedFifth.h"
#include "base.h"
#include <iostream>
        DerivedFifth::DerivedFifth(Base* parent, string name, int status) :
Base(parent, name, status) {};
```

Файл DerivedFifth.h

```
#ifndef DerivedFifth_h
#define DerivedFifth_h
#include "base.h"
#include <iostream>
        class DerivedFifth : public Base {
                public:
                        DerivedFifth(Base* parent, string name, int status);
};
#endif
```

Файл DerivedFourth.cpp

```
#include "DerivedFourth.h"
#include "base.h"
#include <iostream>
        DerivedFourth::DerivedFourth(Base* parent, string name, int status) :
Base(parent, name, status) {};
```

Файл DerivedFourth.h

```
#ifndef DerivedFourth_h
#define DerivedFourth_h
#include "base.h"
#include <iostream>
        class DerivedFourth : public Base {
                public:
                        DerivedFourth(Base* parent, string name, int status);
};
#endif
```

Файл Derived.h

```
#ifndef Derived_h
#define Derived_h
#include "base.h"
#include <iostream>
```

```
        class Derived : public Base {
                public:
                        Derived(Base* parent, string name, int status);
};
#endif
```

Файл DerivedSecond.cpp

```
#include "DerivedSecond.h"
#include "base.h"
#include <iostream>
        DerivedSecond::DerivedSecond(Base* parent, string name, int status) :
Base(parent, name, status) {};
```

Файл DerivedSecond.h

```
#ifndef DerivedSecond_h
#define DerivedSecond_h
#include "base.h"
#include <iostream>
        class DerivedSecond : public Base {
                public:
                        DerivedSecond(Base* parent, string name, int status);
};
#endif
```

Файл DerivedThird.cpp

```
#include "DerivedThird.h"
#include "base.h"
#include <iostream>
        DerivedThird::DerivedThird(Base* parent, string name, int status) :
Base(parent, name, status) {};
```

Файл DerivedThird.h

```
#ifndef DerivedThird_h
#define DerivedThird_h
#include "base.h"
#include <iostream>
        class DerivedThird : public Base {
                public:
                        DerivedThird(Base* parent, string name, int status);
};
#endif
```

Файл main.cpp

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

#include "application.h"
int main() {
        Application app(nullptr);
        app.buildTree();
        app.execute();
}
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
a endtree //	Object tree a	Object tree a
c endtree //	Object tree c	Object tree c

d endtree // Object tree d Object tree d