|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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** | | Кузнецов Владислав Евгеньевич | | | | | | | | |  |  |
|  | | (ФИО, учебная группа) | | | | | | | | |  | (подпись студента) |
|  | | | |  | | | | |  | | |  |
| **Руководитель курсовой работы** | | | | | | Путуридзе Зураб Шотаевич, доцент | | | | |  |  |
|  | | | | | | (должность, звание, ученая степень, ФИО) | | | | |  | (подпись руководителя) |
|  | | | |  | | | | |  | | |  |
| **Рецензент** (при наличии) | | | | | |  | | | | |  |  |
|  | | | | | | (должность, звание, ученая степень, ФИО) | | | | |  | (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 | Кузнецов Владислав Евгеньевич | | Группа | ИВБО-07-20 | |
| **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** | |  | |  | | | | | ( | | Путуридзе З.Ш. | | ) |
|  |  | | | *Подпись руководителя* | | | | |  | | *Ф.И.О. руководителя* | |  |
|  |  | | |  | |  | | | | |  | | |
| **Задание на курсовую работу получил** | | | |  | | | | «25» февраля 2021г | | | |  | |
|  | | |  | | | | ( | | | Кузнецов В.Е. | | ) | |
|  | | | *Подпись обучающегося* | | | |  | | | Ф.И.О. исполнителя | |  | |

Москва 2021 г.

Содержание

[Содержание 3](#_Toc73921973)

[Введение 5](#_Toc73921974)

[Постановка Задачи 6](#_Toc73921975)

[Определение указателя на объект по его координате 6](#_Toc73921976)

[Описание входных данных 7](#_Toc73921977)

[Описание выходных данных 7](#_Toc73921978)

[Метод Решения 8](#_Toc73921979)

[Описание Алгоритма 10](#_Toc73921980)

[Блок-схема Алгоритма 19](#_Toc73921981)

[Код Программы 32](#_Toc73921982)

[Файл App.cpp 32](#_Toc73921983)

[Файл App.h 32](#_Toc73921984)

[Файл Base.cpp 33](#_Toc73921985)

[Файл Base.h 35](#_Toc73921986)

[Файл main.cpp 36](#_Toc73921987)

[Тестирование 37](#_Toc73921988)

[Заключение 38](#_Toc73921989)

[Список Используемой Литературы 39](#_Toc73921990)

Введение

ООП формирует базу для дальнейшего более глубокого изучения программирования. Язык программирования играет здесь важнейшую роль, поскольку неправильный его выбор может усложнить и замедлить дальнейшее изучение предмета. С++ наиболее удобен для изучения ООП т.к. он имеет крайне высокую скорость исполнения кода, а также самый распространенный среди языков, так называемый «Си-образный» синтаксис. Компиляторы С++ имеются на любой платформе, что обеспечивает кроссплатформенность языка. Важно отметить, что чётко разграниченные уровни доступа к членам класса, возможность множественного наследования и динамический полиморфизм позволяют быстро усвоить основные концепции ООП: абстракция, наследование, инкапсуляция и полиморфизм. В дальнейшем, эти знания позволят облегчить и ускорить процесс изучения других объектно-ориентированных языков программирования. С++ используется в программировании микроконтроллеров, в проектировке нейронных сетей, в создании программ для промышленных предприятий, а также персональных компьютеров и иногда мобильных устройств. Немаловажно, к тому же, что этот язык активно обновляется и поддерживается.

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

Используемые классы:

1. list - класс двухсвязных списков стандартной библиотеки. Используемые методы:
   1. .empty() - проверяет является ли контейнер стека пустым;
   2. .find() - выполняет поиск первого вхождения заданного значения в контейнере;
   3. .erase() - удаляет из списка один или диапазон элементов.
2. класс App:
3. Свойства:
   1. Base\* tree - приватное поле App.
4. Методы:
   1. Конструктор по умолчанию;
   2. void build\_tree() - метод постройки дерева;
   3. void exec\_app() - метод запуска системы.
5. Используем класс Base:
   1. Свойства:

protected:

* + 1. string name - поле объекта;
    2. int status - целый статус объекта;
    3. Base\* parent - указатель на родителя объекта;
    4. list children - список детей.
  1. Методы:
     1. Параметризированный конструктор Base (Base\* parent, string name, int status);
     2. Base\* find\_node (string name) - метод поиска узла дерева по имени;
     3. Base\* get\_node (string way) - метод получения указателя на узел дерева по координате way;
     4. void input\_tree() - метод ввода дерева;
     5. void print\_nodes\_status() - метод вывода списка готовности объектов дерева;
     6. void print\_tree(int level) - метод вывода имён объектов дерева.

1. Используем классы Child\_2, Child\_3, Child\_4, Child\_5, Child\_6:
   1. Свойства: наследуемые от Base.
   2. Методы:
      1. наследуемые от Base;
      2. Параметризированные конструкторы Child\_(2/3/4/5/6) (Base\* parent, string name, int status).

Таблица 1. «Описание иерархии наследования классов»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Имя класса** | **Классы-Наследники** | **Модификатор доступа при наследовании** | **Описание** | **Номер** |
| 1 | Base |  |  | Базовый класс в иерархии классов. Содержит основные поля и методы |  |
| App | public |  | 1 |
| Child\_2 | public |  | 2 |
| Child\_3 | public |  | 3 |
| Child\_4 | public |  | 4 |
| Child\_5 | public |  | 5 |
| Child\_6 | public |  | 6 |
| 2 | App |  |  | Класс корневого объекта (Приложения) |  |
| 3 | Child\_2, Child\_3, Child\_4, Child\_5, Child\_6 |  |  | Класс объектов, подчиненных корневому объекту класса App. |  |

Описание Алгоритма

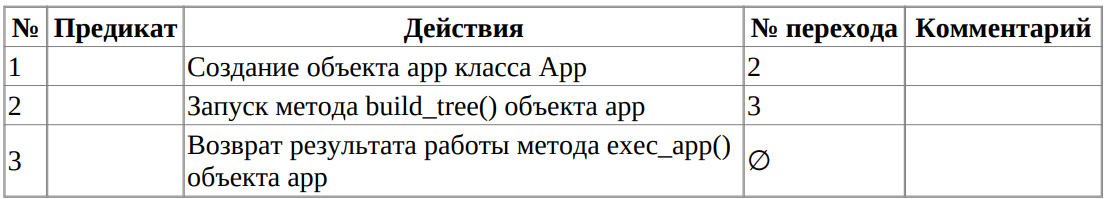
Функция: main

Функционал: Точка входа в программу

Параметры: без параметров

Возвращаемое значение: int 0

Таблица 2. «Функция main»



Класс объекта: App

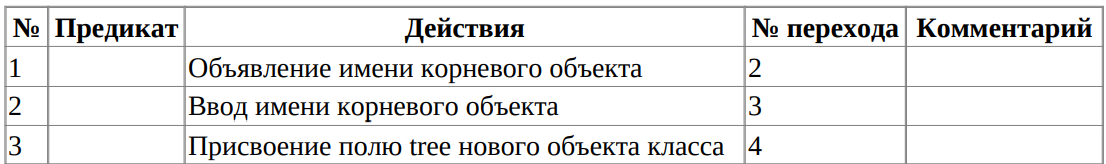
Модификатор доступа: publicМетод: build\_tree

Функционал: Построение дерева объектов

Параметры: без параметров

Возвращаемое значение: без возвращаемого значения

Таблица 3. «Метод build\_tree»





Класс объекта: App

Модификатор доступа: public

Метод: exec\_app

Функционал: метод запуска программы

Параметры: без параметров

Возвращаемое значение: int 0

Таблица 4. «Метод exec\_app»



Класс объекта: Base

Модификатор доступа: public

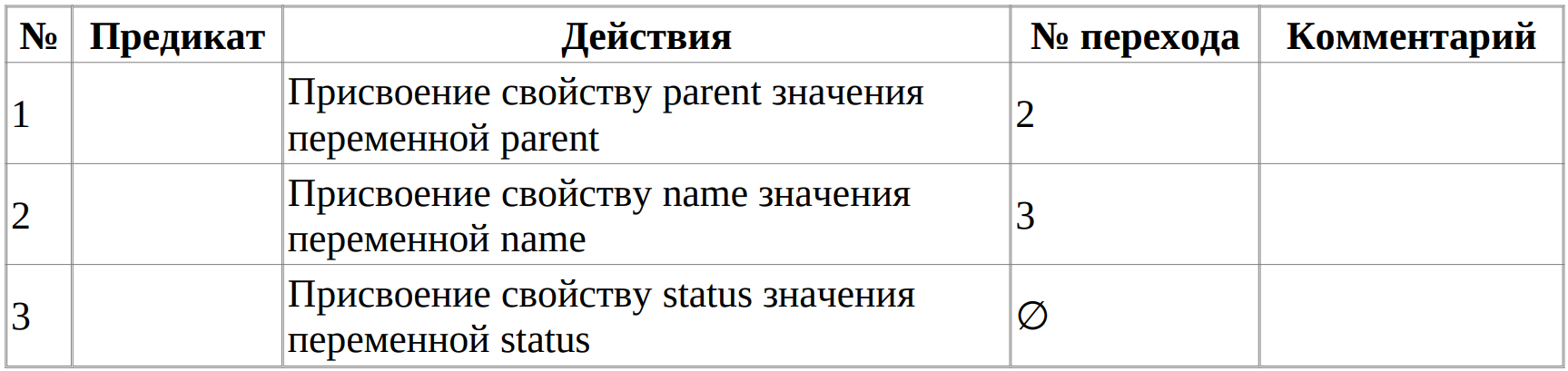
Метод: Base

Функционал: Параметризированный конструктор

Параметры: Base \*parent, string name, int status

Возвращаемое значение: нет

Таблица 5. «Метод Base»



Класс объекта: Base

Модификатор доступа: public

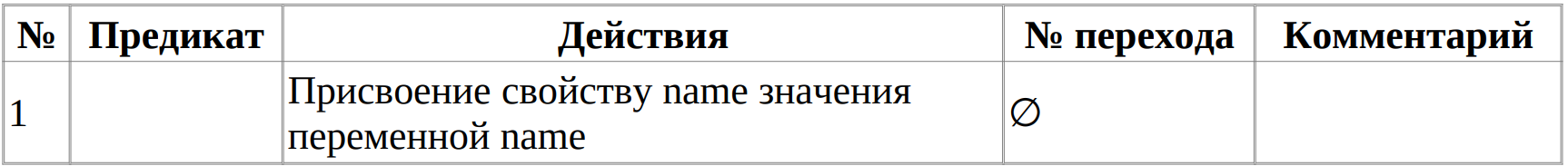
Метод: set\_Name

Функционал: Метод определения имени объекта

Параметры: string name

Возвращаемое значение: 0

Таблица 6. «Метод set\_Name»



Класс объекта: Base

Модификатор доступа: public

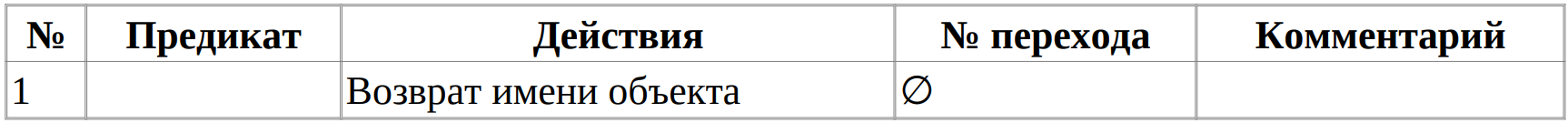
Метод: get\_Name

Функционал: Метод получения имени объекта

Параметры: нет

Возвращаемое значение: name

Таблица 7. «Метод get\_Name»



Класс объекта: Base

Модификатор доступа: public

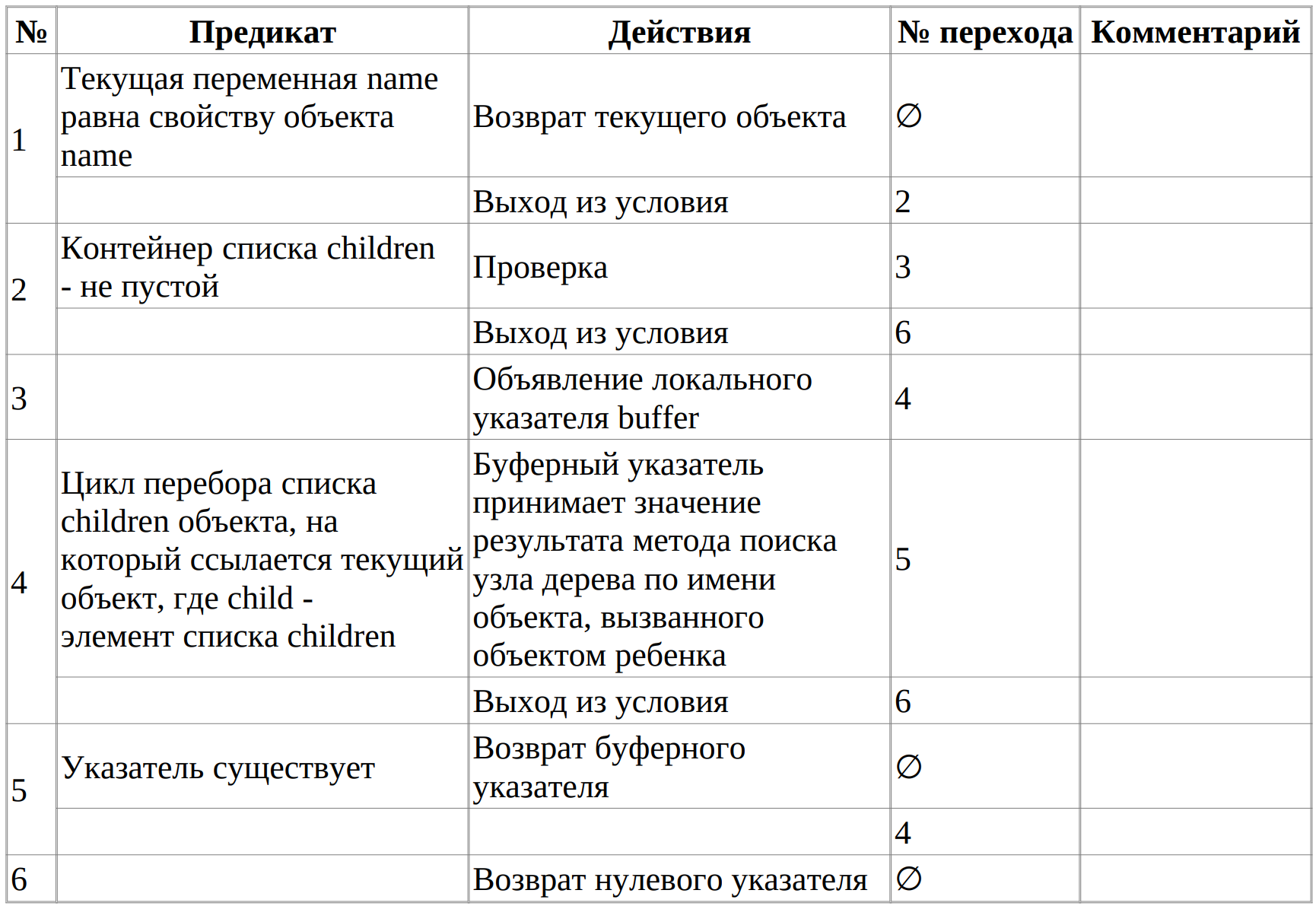
Метод: find\_node

Функционал: метод поиска узла по имени

Параметры: string name

Возвращаемое значение:int 0

Таблица 8. «Метод find\_node»



Класс объекта: Base

Модификатор доступа: public

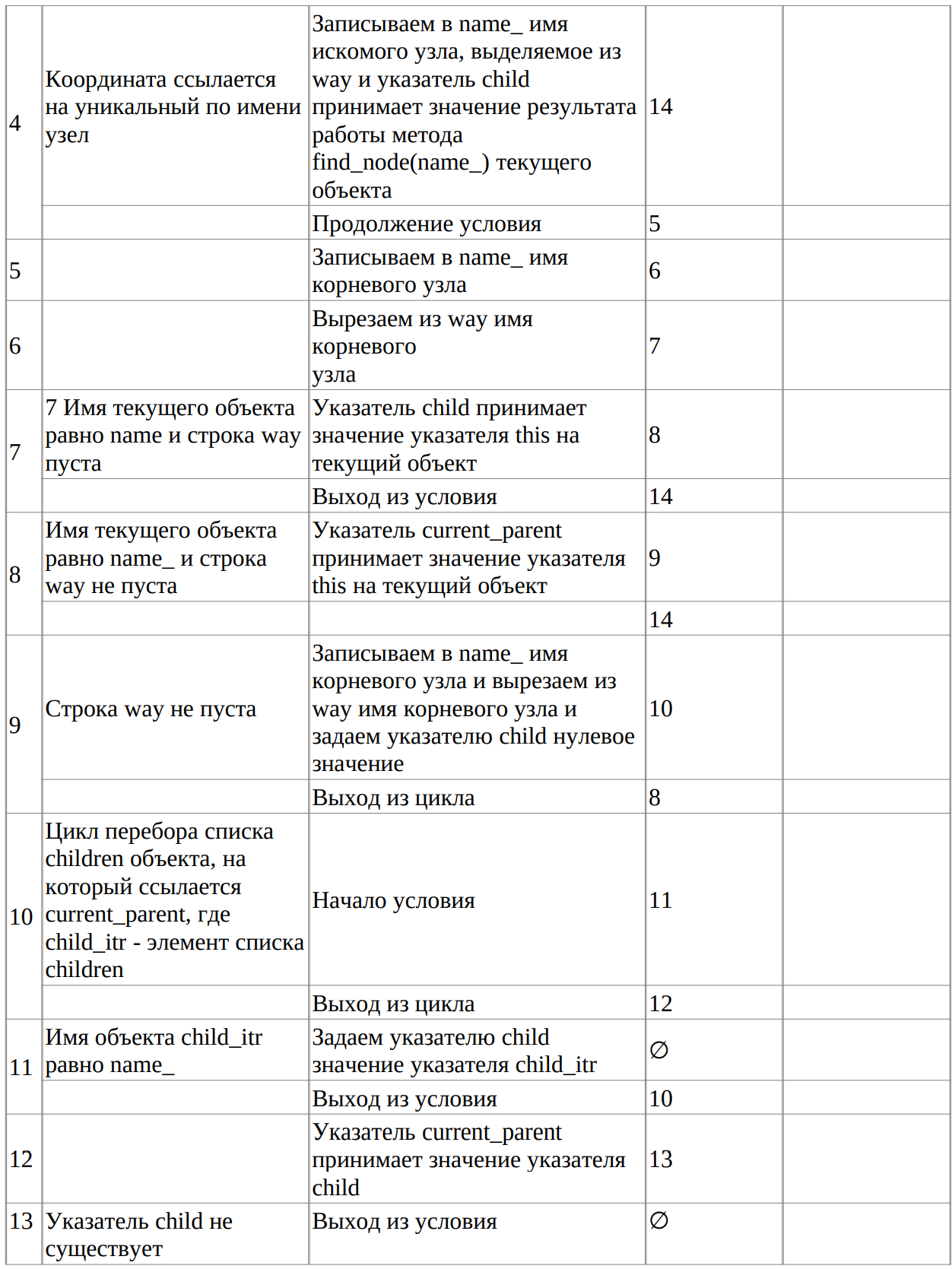
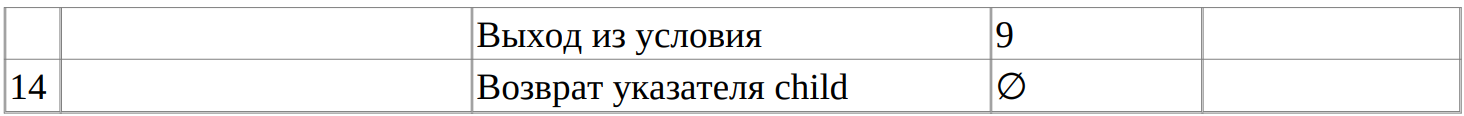
Метод: get\_node

Функционал: метод получения указателя на узел дерева по координате way

Параметры: string way

Возвращаемое значение: Base\*

Таблица 9. «Метод get\_node»



Класс объекта: Base

Модификатор доступа: public

Метод: input\_tree

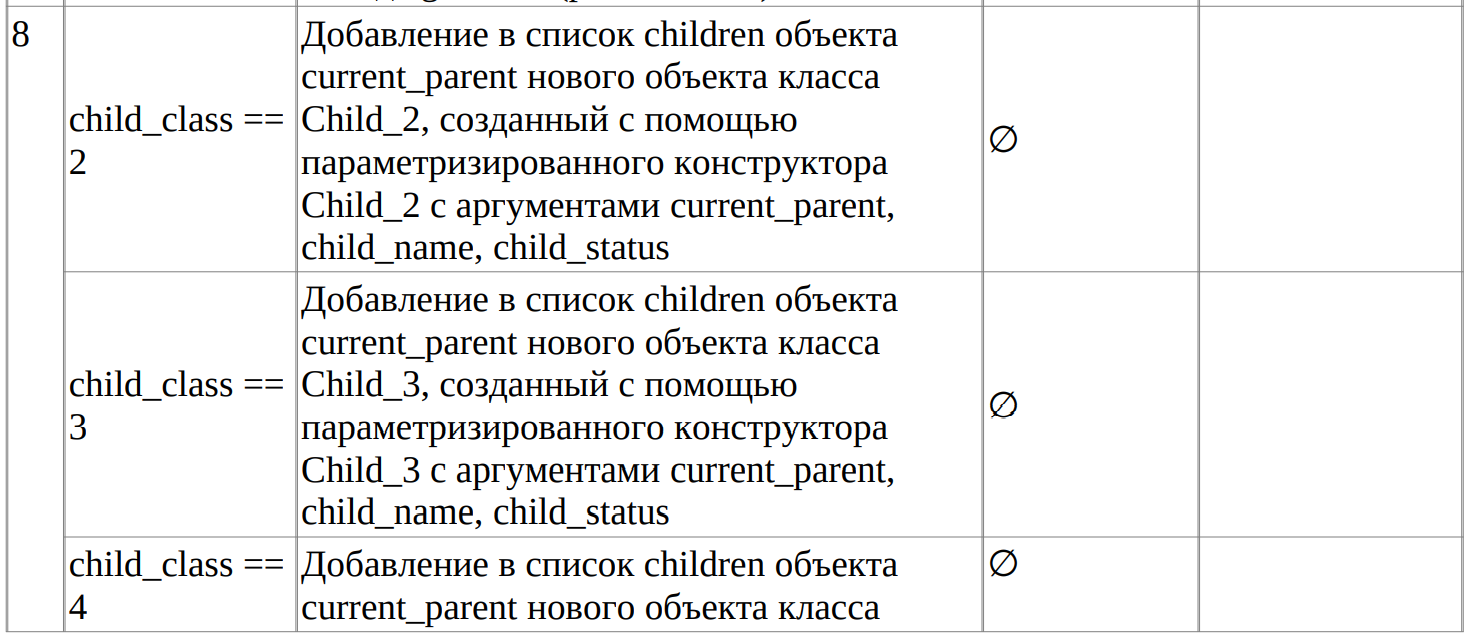
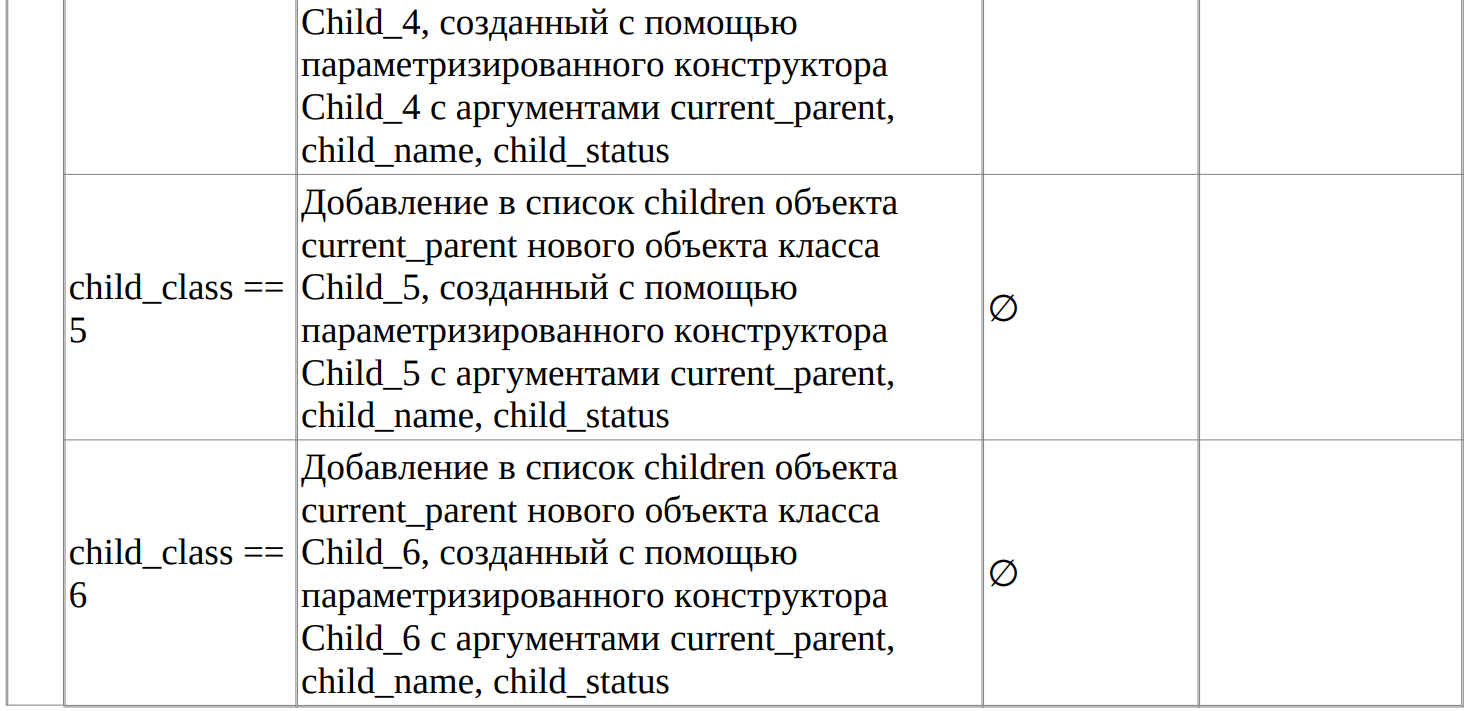
Функционал: метод ввода объектов дерева

Параметры: нет

Возвращаемое значение: int 0

Таблица 10. «Метод input\_tree»





Класс объекта: Base

Модификатор доступа: public

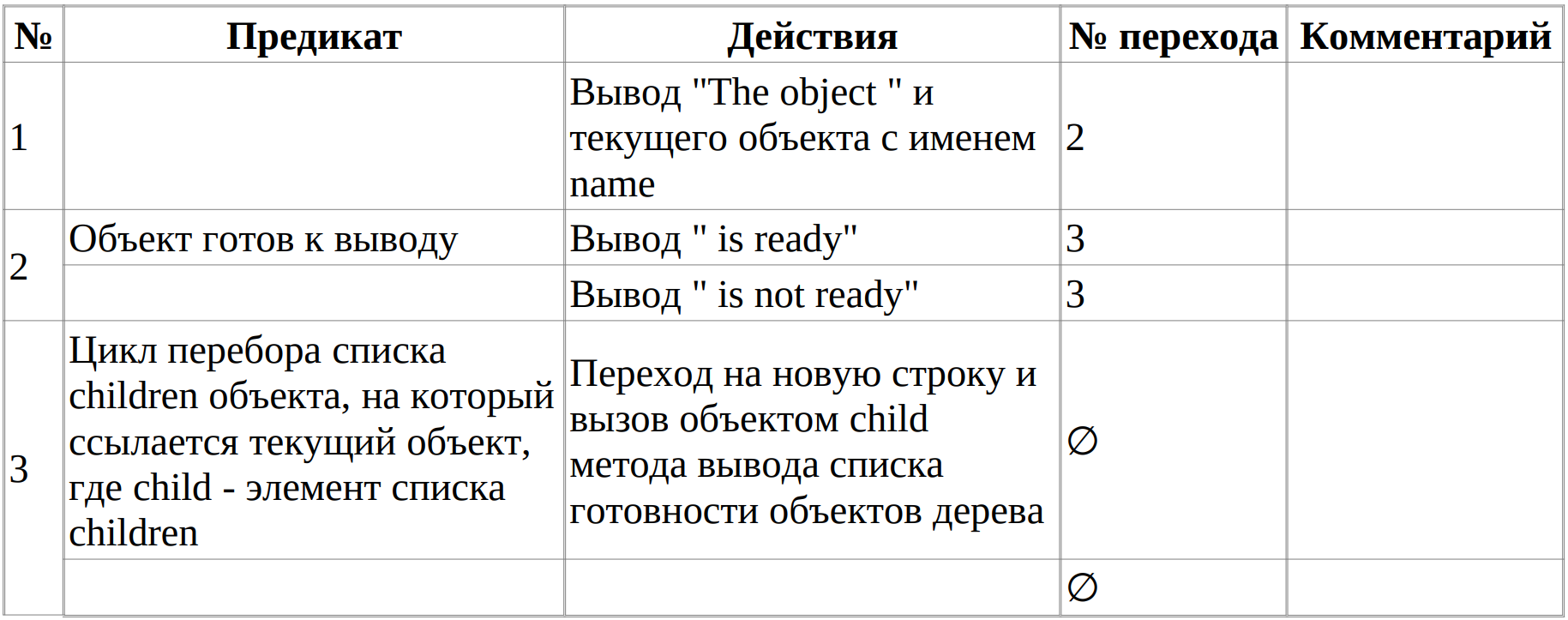
Метод: print\_nodes\_status

Функционал: метод вывода списка готовности объектов дерева

Параметры: нет

Возвращаемое значение: int 0

Таблица 11. «Метод print\_nodes\_status»



Класс объекта: Base

Модификатор доступа: public

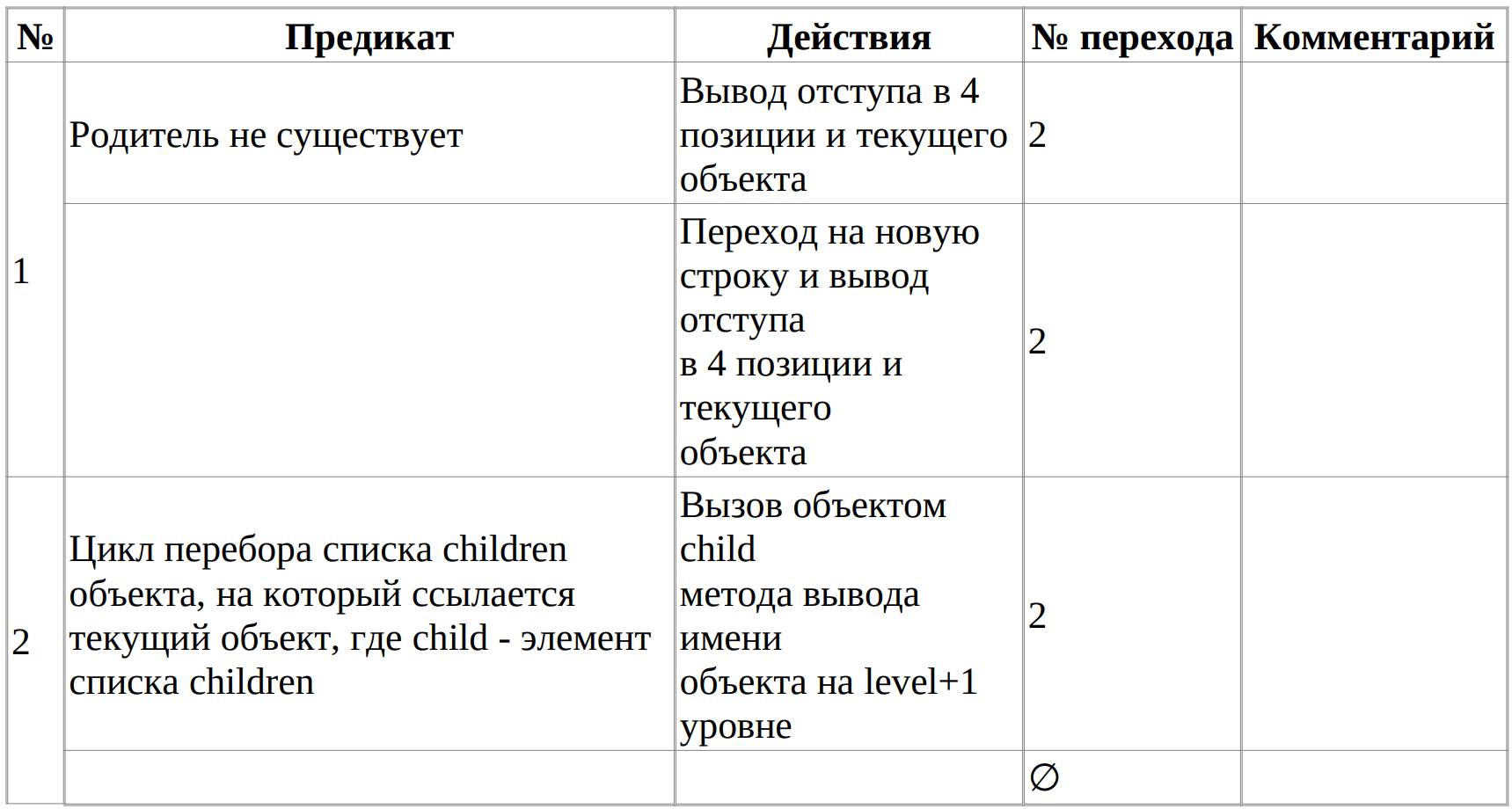
Метод: print\_tree

Функционал: метод вывода имён объектов дерева

Параметры: int level

Возвращаемое значение: int 0

Таблица 12. «Метод print\_tree »



Блок-схема Алгоритма

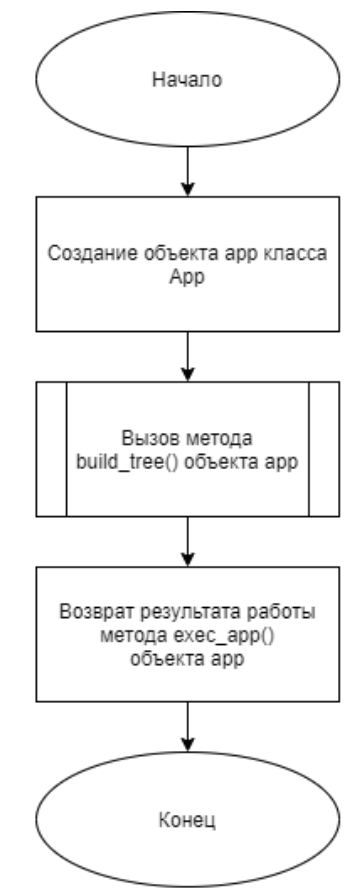


Рис.1 Блок-схема функции main

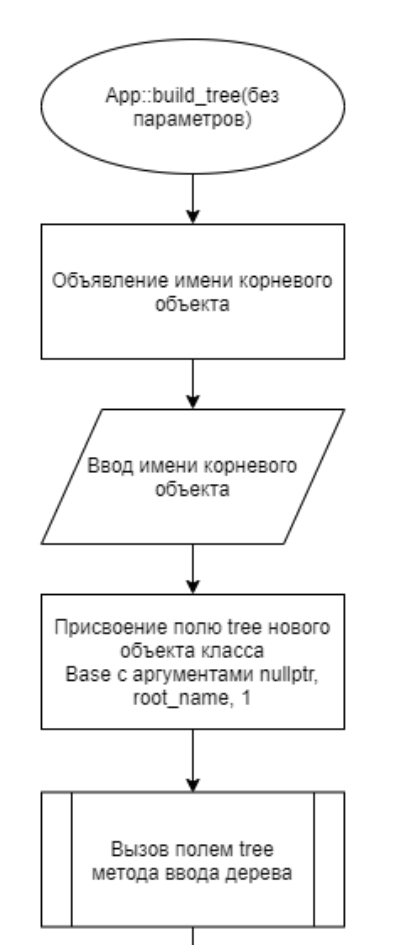


Рис.2 Блок-схема метода App::build\_tree()

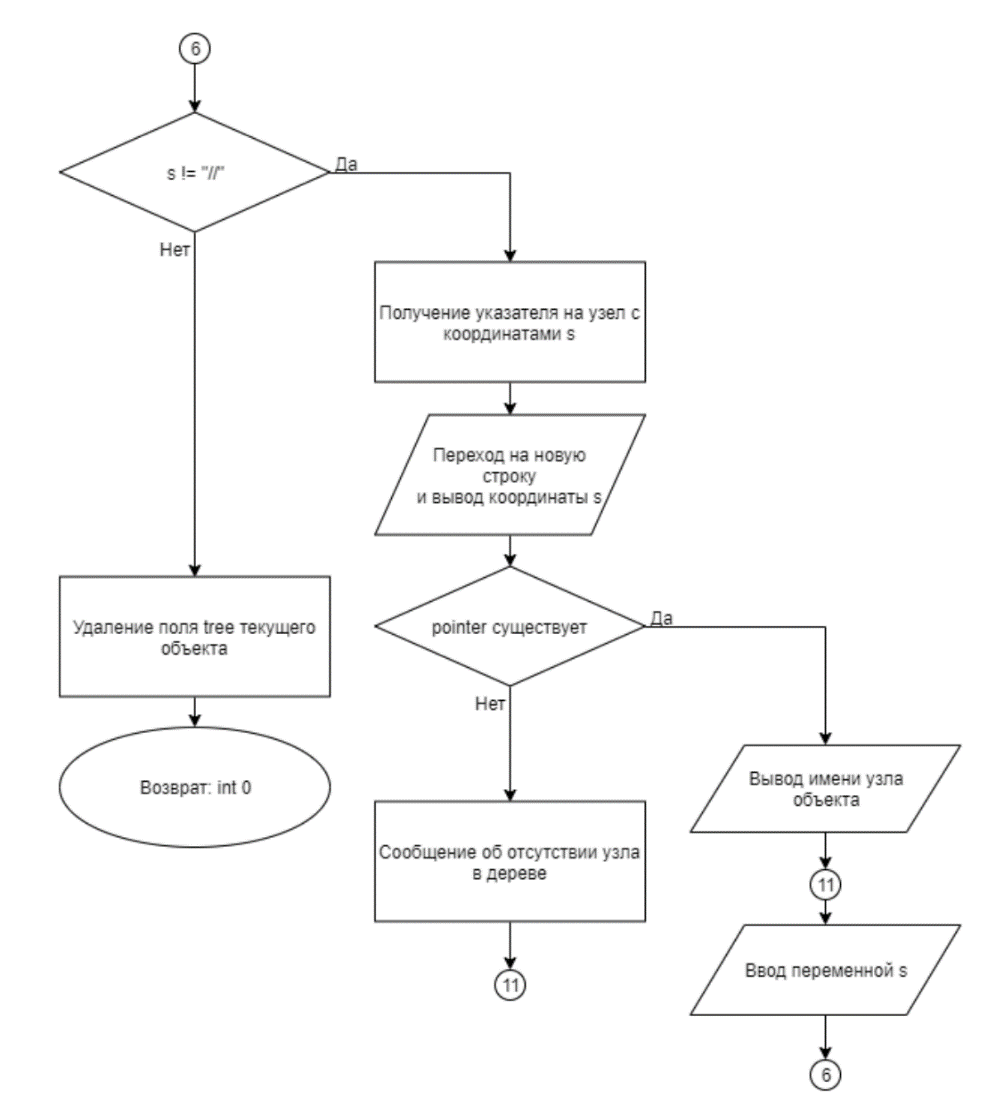
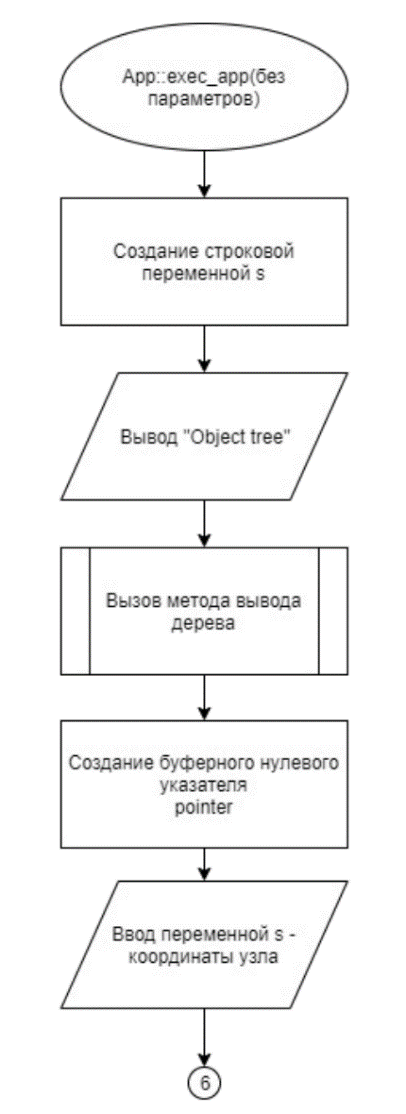


Рис.3 Блок-схема метода App::exec\_app()

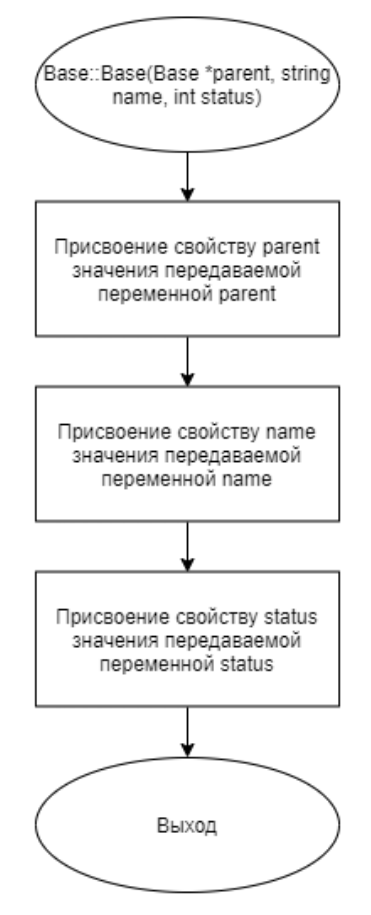


Рис.4 Блок-схема метода Base::Base(Base \*parent, string name, int status)

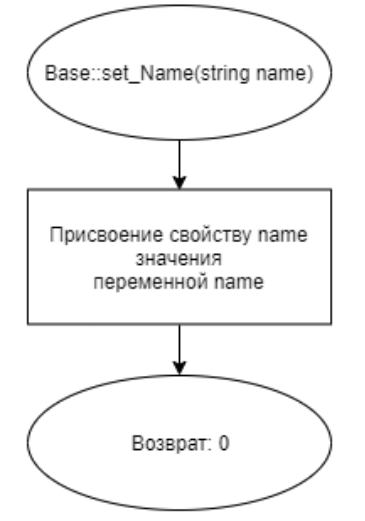


Рис.5 Блок-схема метода Base::set\_Name(string name)

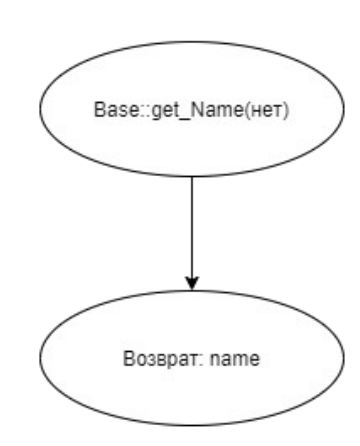
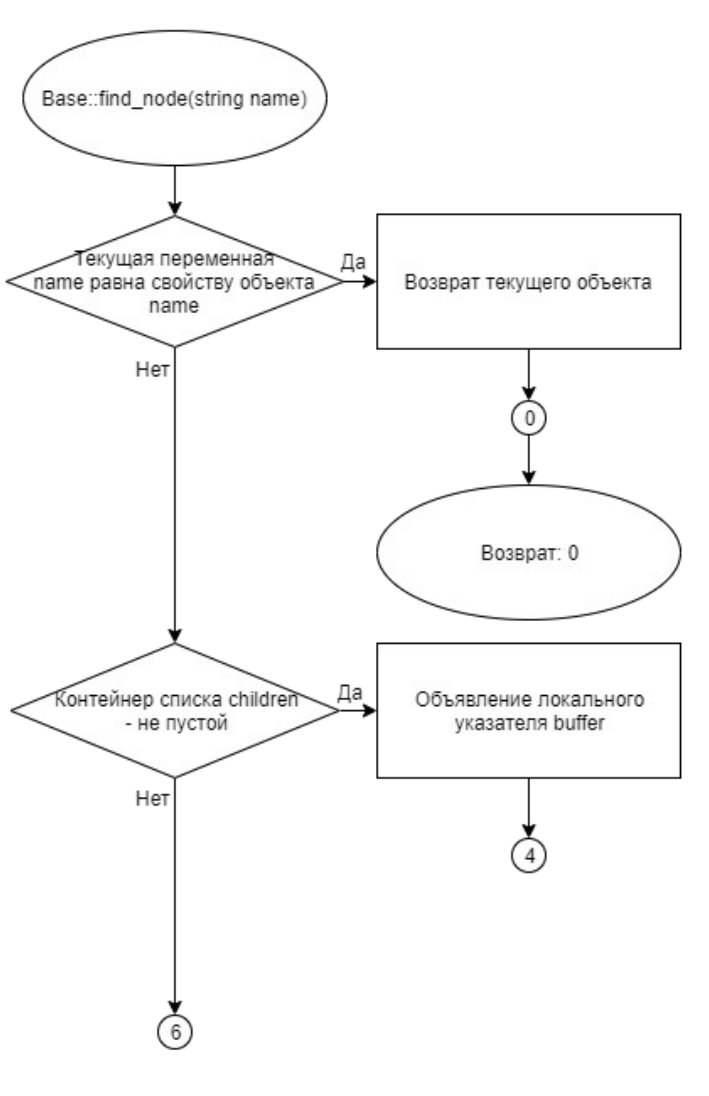


Рис.6 Блок-схема метода Base::get\_Name()



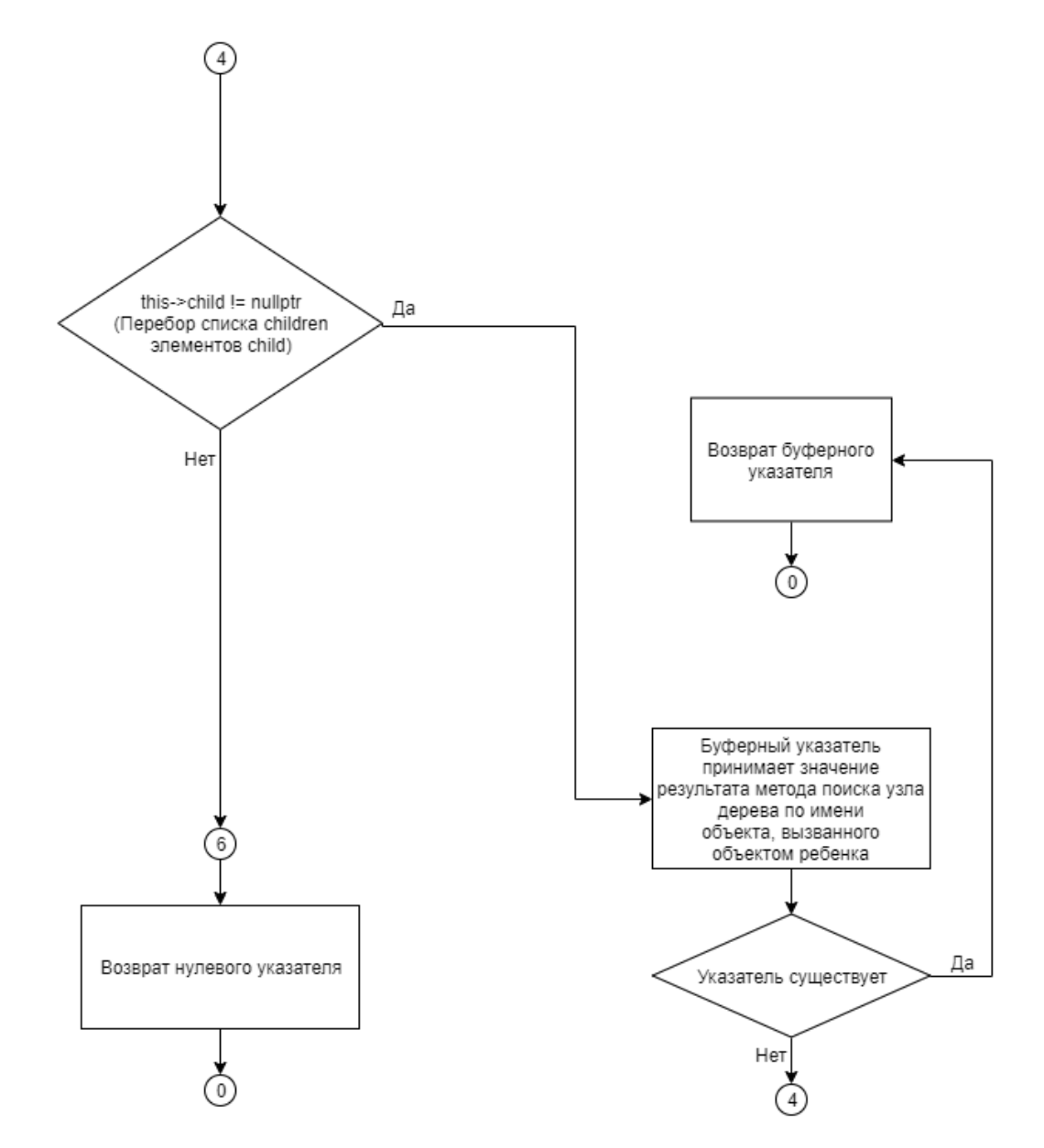


Рис.7 Блок-схема метода Base::find\_node(string name)

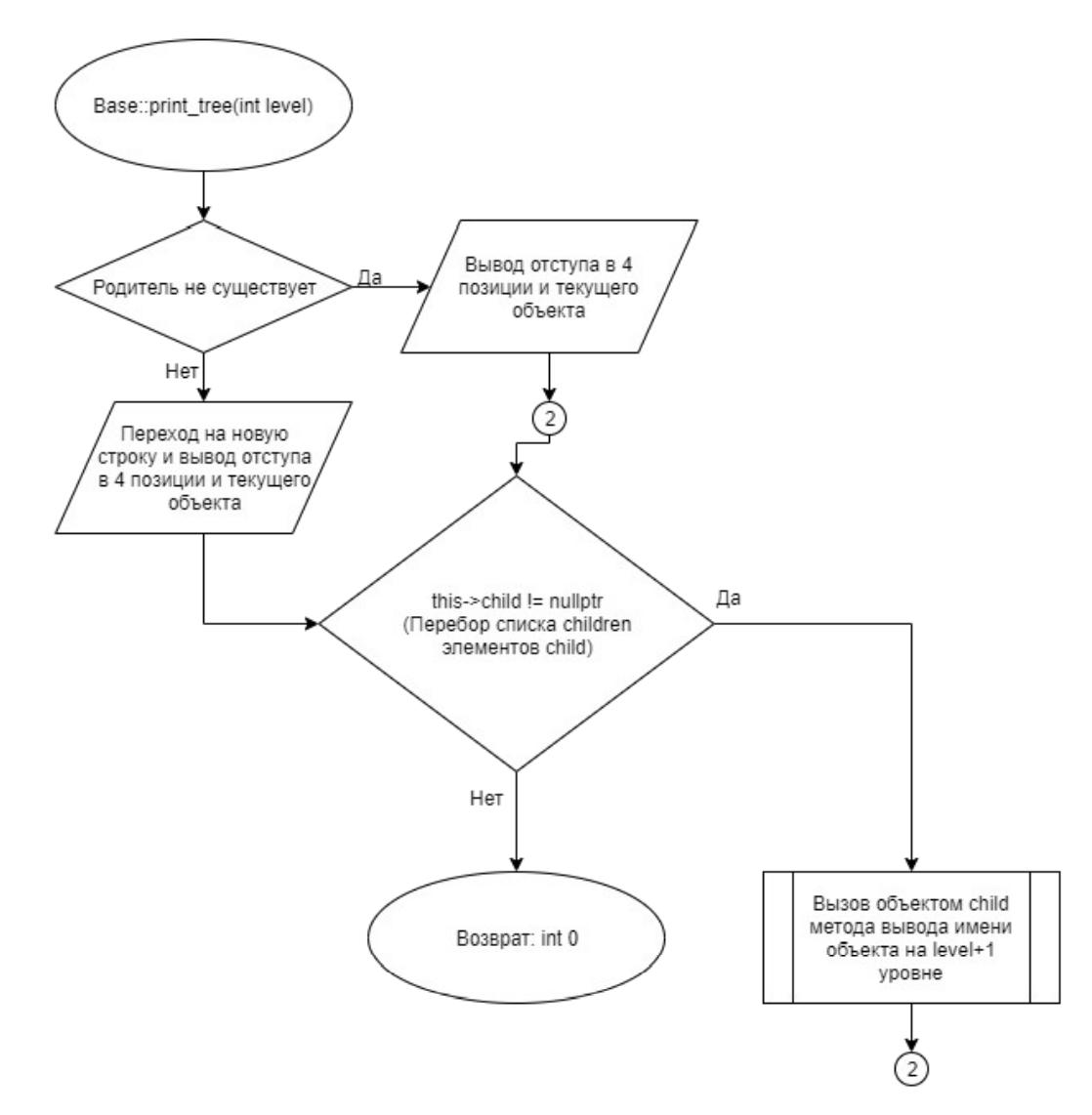


Рис.8 Блок-схема метода Base::print\_tree(int level)

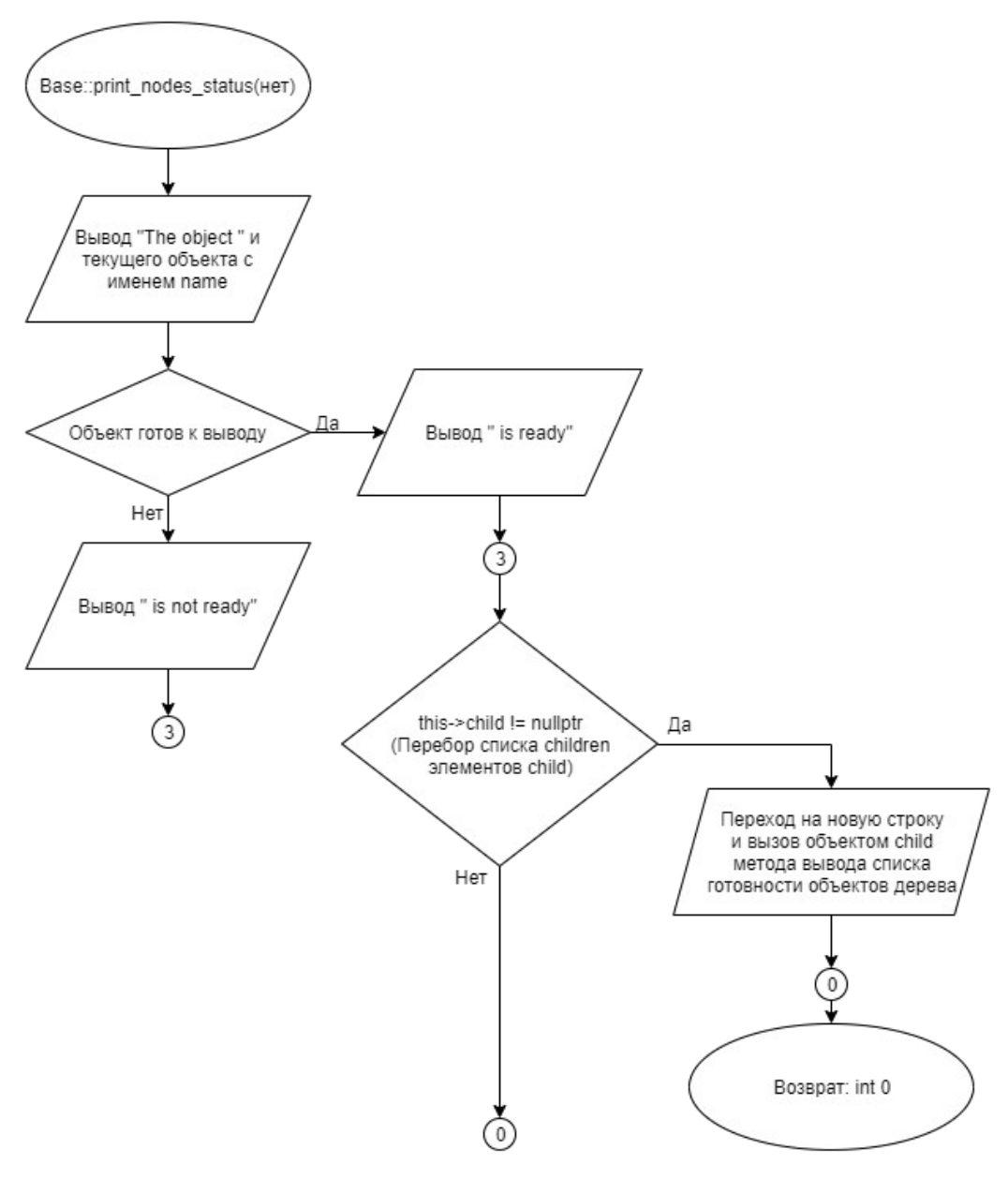


Рис.9 Блок-схема метода Base::print\_nodes\_status()

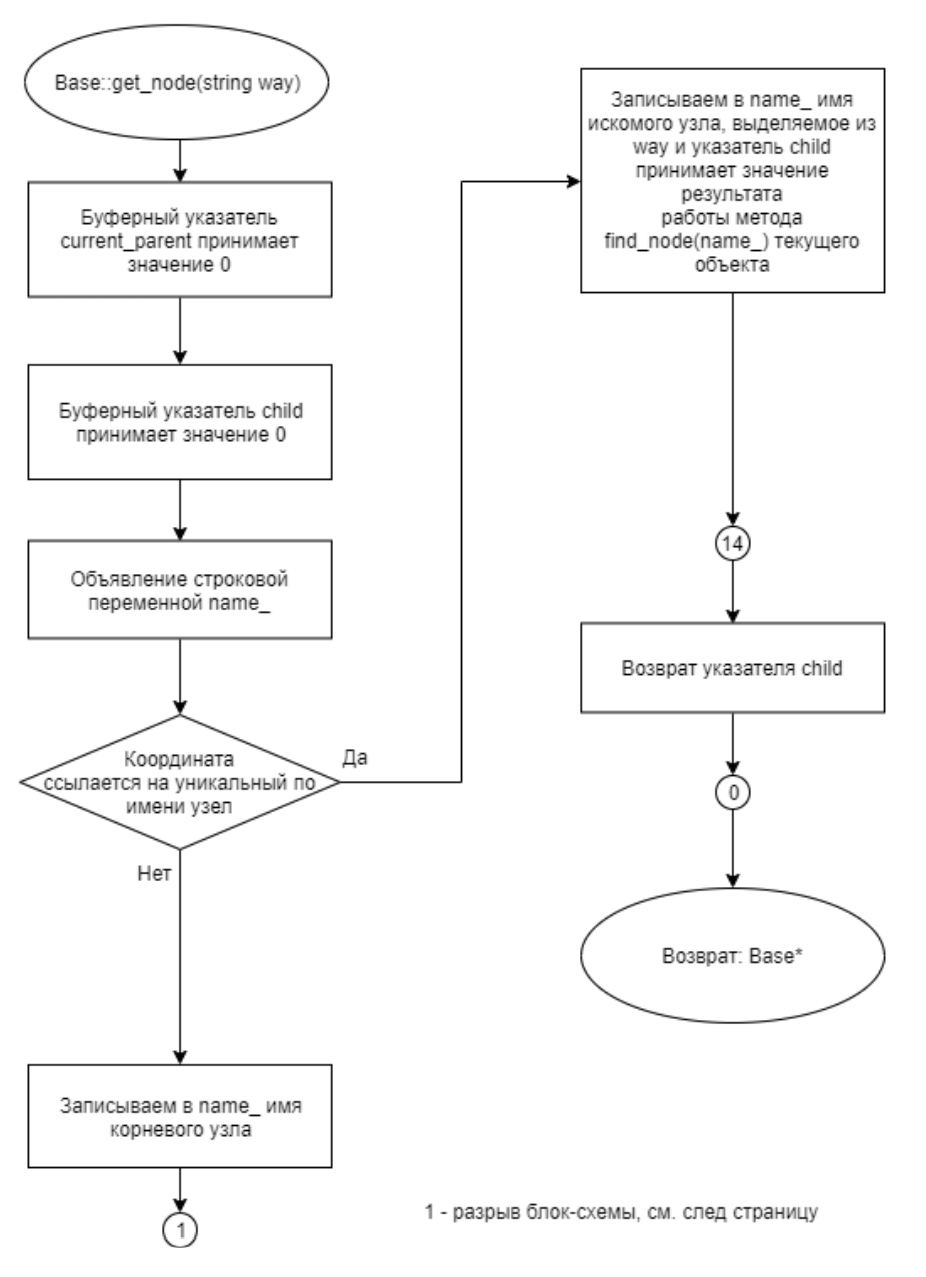


Рис. 10.1 Часть блок-схемы метода Base::get\_node(string\_way)

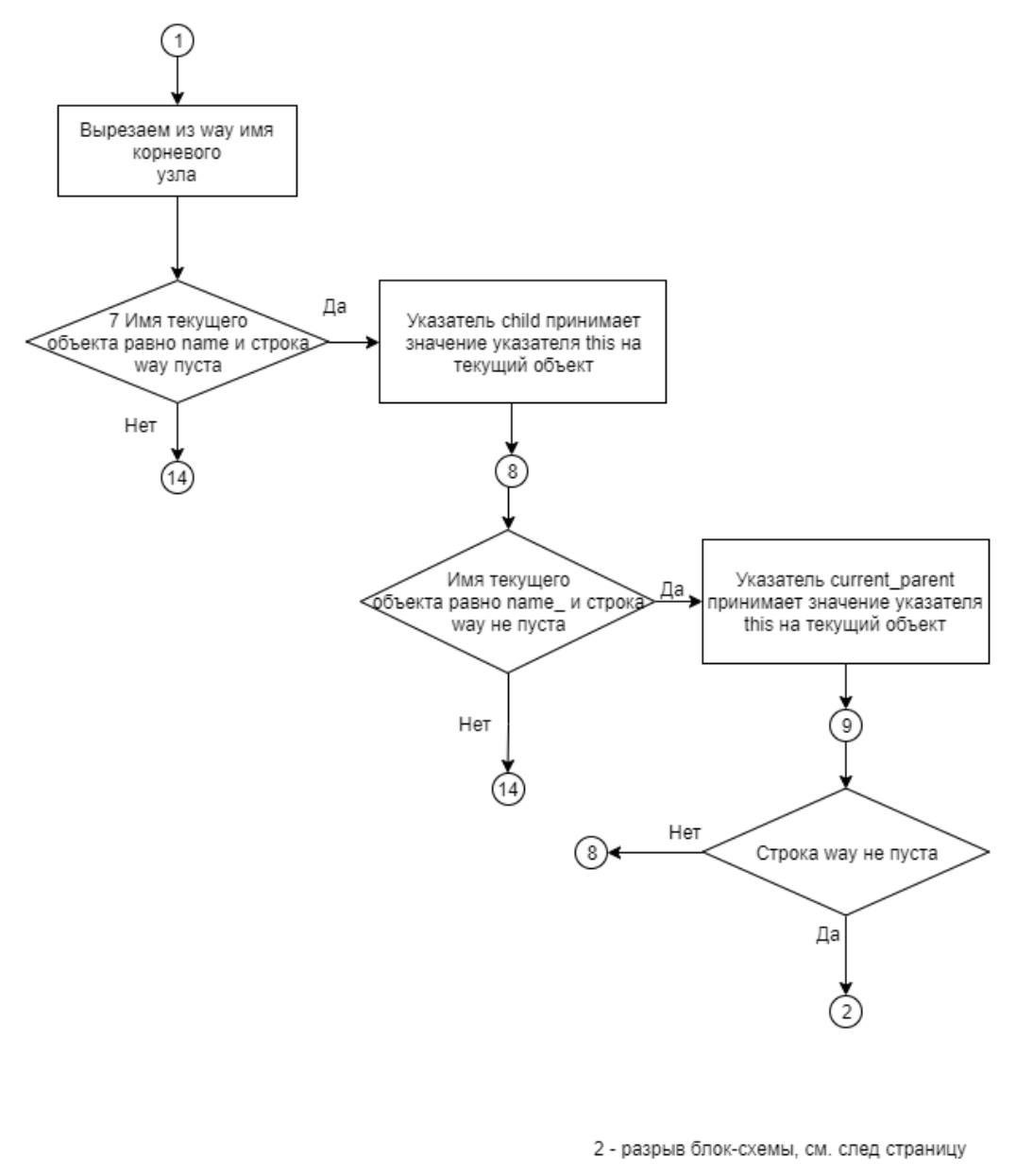


Рис. 10.2 Часть блок-схемы метода Base::get\_node(string\_way)

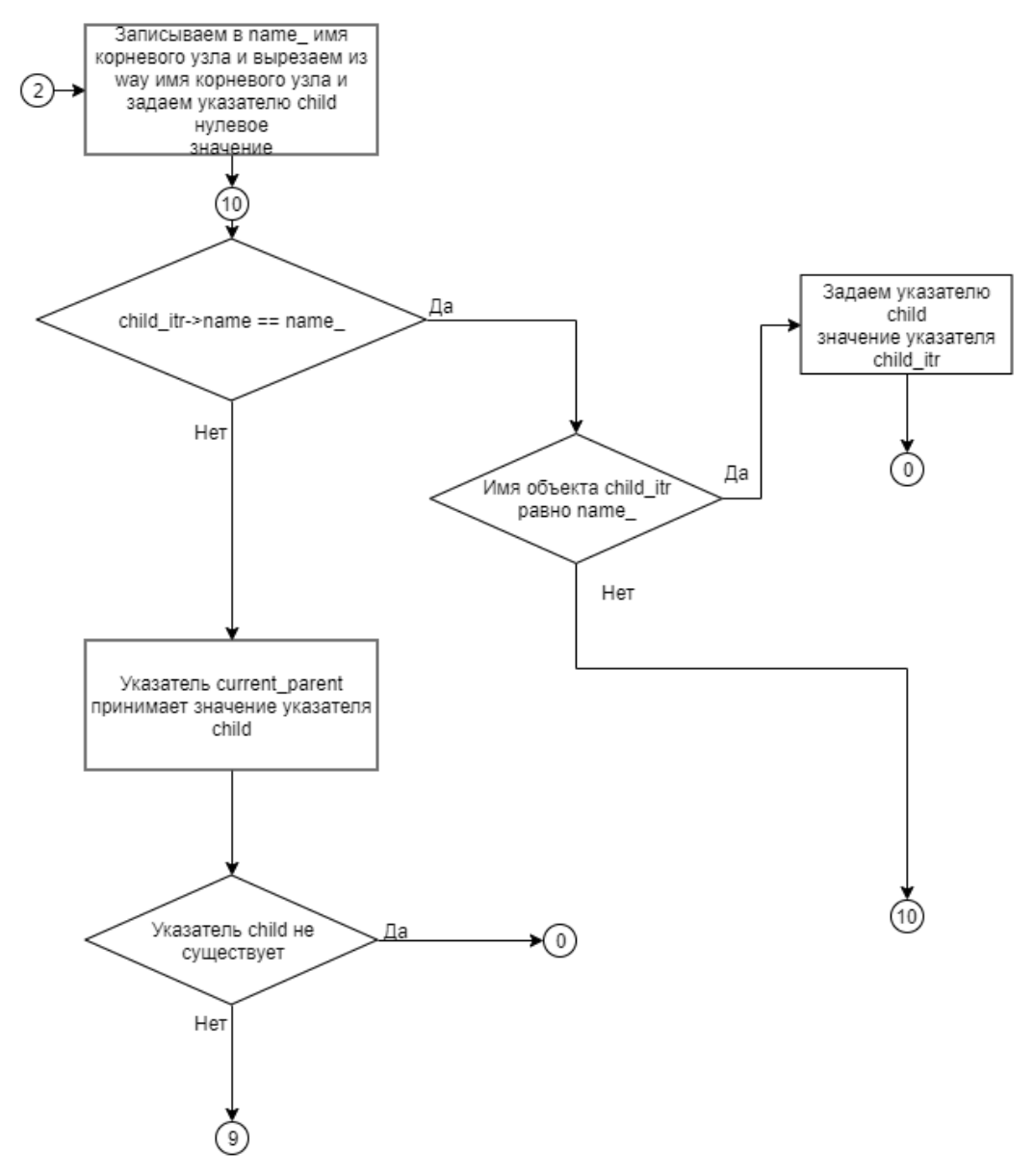


Рис. 10.3 Часть блок-схемы метода Base::get\_node(string\_way)

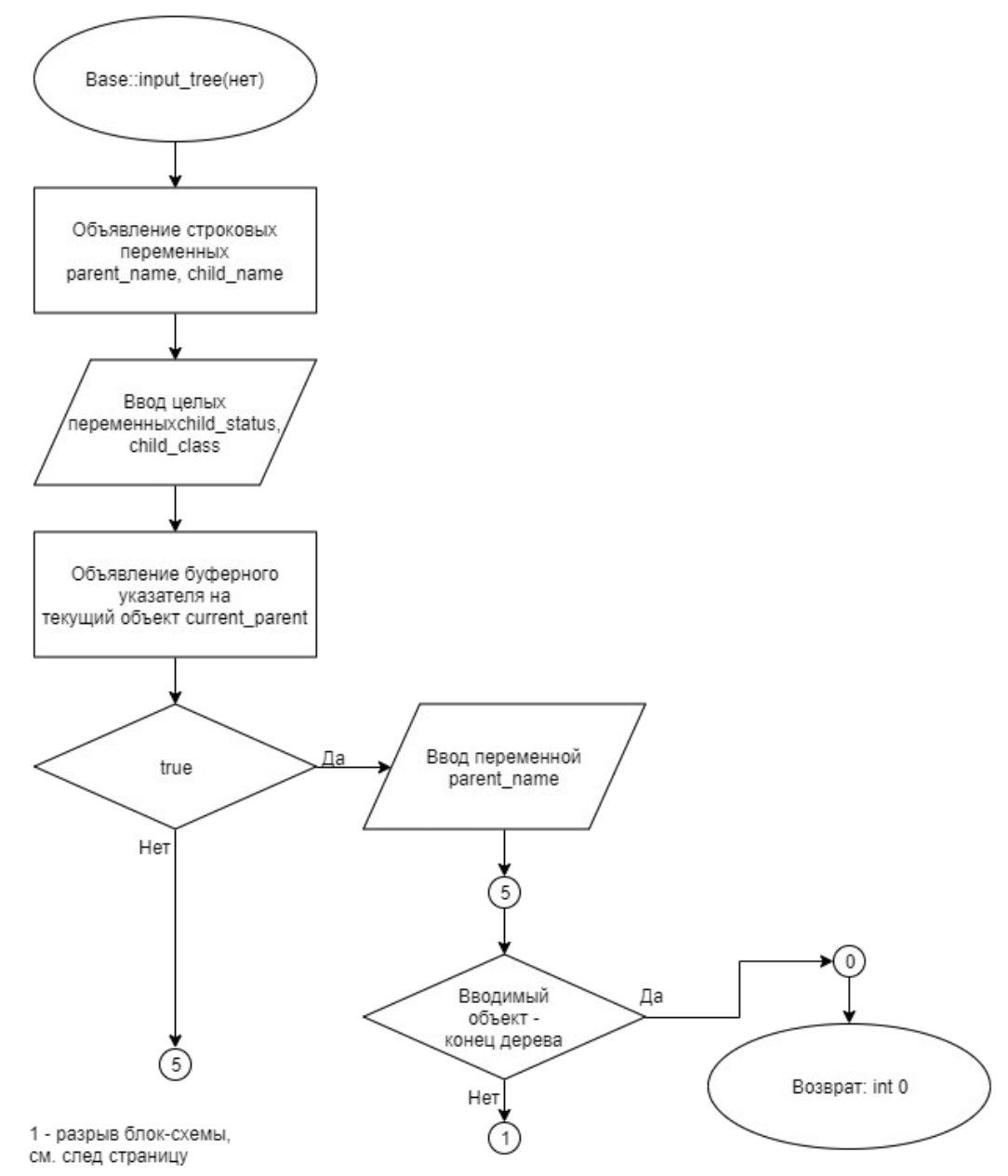


Рис. 11.1 Часть блок-схемы метода Base::input\_tree()

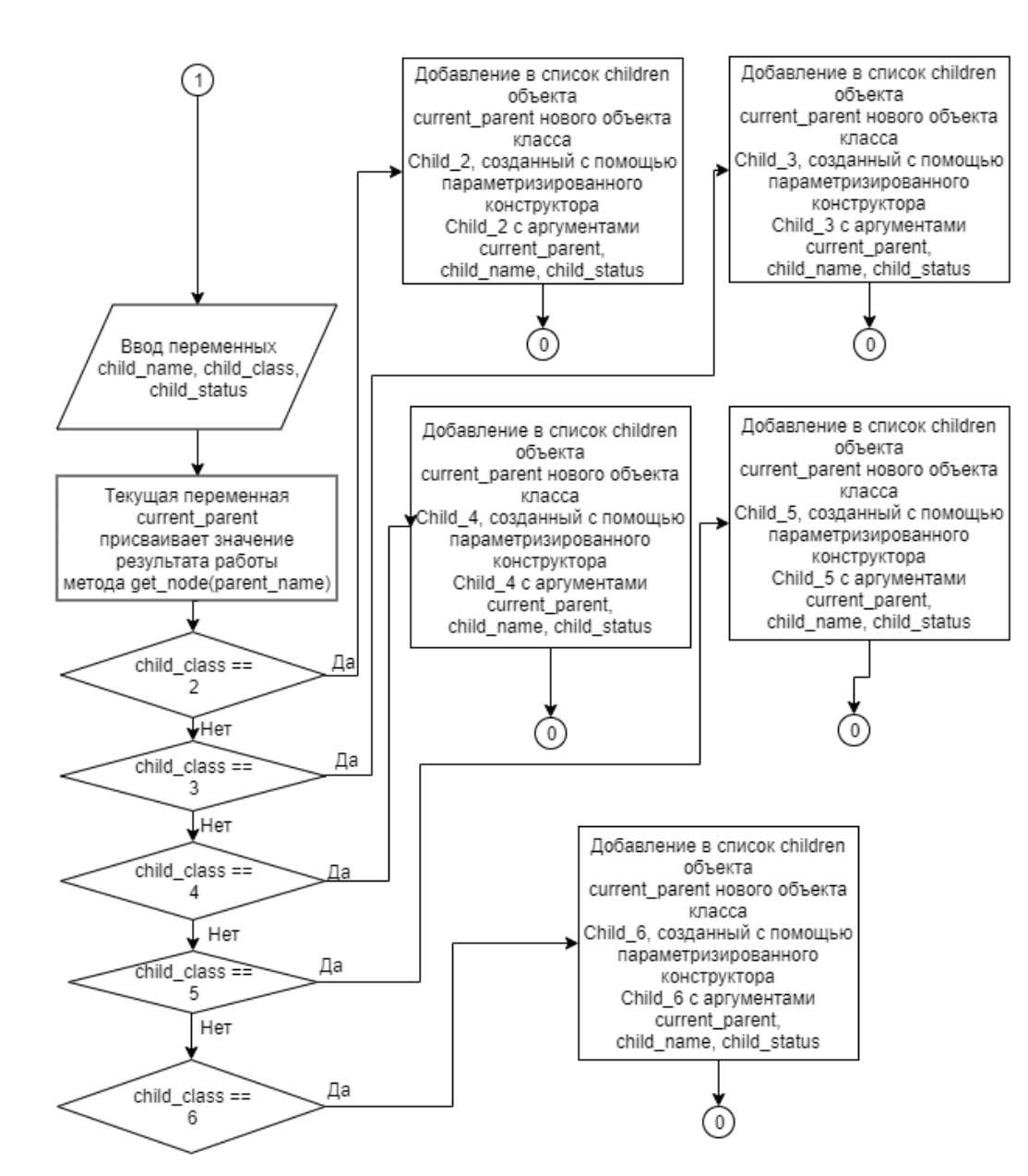


Рис. 11.2 Часть блок-схемы метода Base::input\_tree()

Код Программы

Файл App.cpp

include "App.h"

void App::build\_tree()

{

string root\_name;

cin >> root\_name;

this->tree = new Base(nullptr, root\_name, 1);

this->tree->input\_tree();

return;

}

int App::exec\_app()

{

string s;

cout << "Object tree" << endl;

this->tree->print\_tree();

Base \*pointer = nullptr;

cin >> s;

while (s != "//")

{

pointer = this->tree->get\_node(s);

cout << endl << s;

if (pointer)

cout << " Object name: " << pointer->get\_name();

else

cout << " Object not found";

cin >> s;

}

delete this->tree;

return 0;

}

Файл App.h

#ifndef App\_h

#define App\_h

#include "Base.h"

#include <iostream>

class App

{

private:

Base \*tree = nullptr;

public:

void build\_tree();

int exec\_app();

};

#endif

Файл Base.cpp

#include "Base.h"

#include <string>

Base::Base(Base \*parent, string name, int status)

{

this->parent = parent;

this->name = name;

this->status = status;

}

void Base::set\_Name(string name)

{

this->name = name;

}

string Base::get\_name()

{

return this->name;

}

Base \*Base::find\_node(string name)

{

if (name == this->name)

{

return this;

}

else if (!this->children.empty())

{

Base \*buffer;

for (auto child : this->children)

{

buffer = child->find\_node(name);

if (buffer)

return buffer;

}

}

return nullptr;

}

Base \*Base::get\_node(string way)

{

Base \*current\_parent = nullptr;

Base \*child = nullptr;

string name\_;

if (way[0] == '/' && way[1] == '/')

{

name\_ = way.substr(2);

child = this->find\_node(name\_);

}

else

{

name\_ = way.substr(1, way.find('/', 1) - 1);

way.erase(0, way.find('/', 1));

if (this->name == name\_ && way == "")

{

child = this;

}

else if (this->name == name\_ && way != "")

{

current\_parent = this;

while (way != "")

{

name\_ = way.substr(1, way.find('/', 1) - 1);

way.erase(0, way.find('/', 1));

child = nullptr;

for (auto child\_itr : current\_parent -> children)

{

if (child\_itr->name == name\_)

{

child = child\_itr;

break;

}

}

current\_parent = child;

if (!child)

break;

}

}

}

return child;

}

void Base::input\_tree()

{

string parent\_name, child\_name;

int child\_status, child\_class;

Base \*current\_parent = this;

while (true)

{

cin >> parent\_name;

if (parent\_name == "endtree")

break;

cin >> child\_name >> child\_class >> child\_status;

current\_parent = this->get\_node(parent\_name);

if (child\_class == 2)

current\_parent->children.push\_back(new Child\_2(current\_parent, child\_name, child\_status));

else if (child\_class == 3)

current\_parent->children.push\_back(new Child\_3(current\_parent, child\_name, child\_status));

else if (child\_class == 4)

current\_parent->children.push\_back(new Child\_4(current\_parent, child\_name, child\_status));

else if (child\_class == 5)

current\_parent->children.push\_back(new Child\_5(current\_parent, child\_name, child\_status));

else if (child\_class == 6)

current\_parent->children.push\_back(new Child\_6(current\_parent, child\_name, child\_status));

}

}

void Base::print\_nodes\_status()

{

cout << "The object " << this->name;

if (this->status > 0)

cout << " is ready";

else

cout << " is not ready";

for (auto child : this->children)

{

cout << endl;

child->print\_nodes\_status();

}

return;

}

void Base::print\_tree(int level)

{

if (this->parent == nullptr)

cout << string(level \* 4, ' ') << this->name;

else

cout << endl << string(level \* 4, ' ') << this->name;

for (auto child : this->children)

child->print\_tree(level + 1);

return;

}

Child\_2::Child\_2(Base \*parent, string name, int status) : Base(parent, name, status){};

Child\_3::Child\_3(Base \*parent, string name, int status) : Base(parent, name, status){};

Child\_4::Child\_4(Base \*parent, string name, int status) : Base(parent, name, status){};

Child\_5::Child\_5(Base \*parent, string name, int status) : Base(parent, name, status){};

Child\_6::Child\_6(Base \*parent, string name, int status) : Base(parent, name, status){};

Файл Base.h

#ifndef Base\_h

#define Base\_h

#include <list>

#include <iostream>

using namespace std;

class Base

{

protected:

string name;

int status;

Base \*parent;

list<Base \*> children;

public:

Base(Base \*parent, string name, int status);

Base \*find\_node(string name);

Base \*get\_node(string way);

void input\_tree();

void print\_nodes\_status();

void print\_tree(int level = 0);

void set\_Name(string name);

string get\_name();

};

class Child\_2 : public Base

{

public:

Child\_2(Base \*parent, string name, int status);

};

class Child\_3 : public Base

{

public:

Child\_3(Base \*parent, string name, int status);

};

class Child\_4 : public Base

{

public:

Child\_4(Base \*parent, string name, int status);

};

class Child\_5 : public Base

{

public:

Child\_5(Base \*parent, string name, int status);

};

class Child\_6 : public Base

{

public:

Child\_6(Base \*parent, string name, int status);

};

#endif

Файл main.cpp

#include "App.h"

using namespace std;

int main()

{

App app;

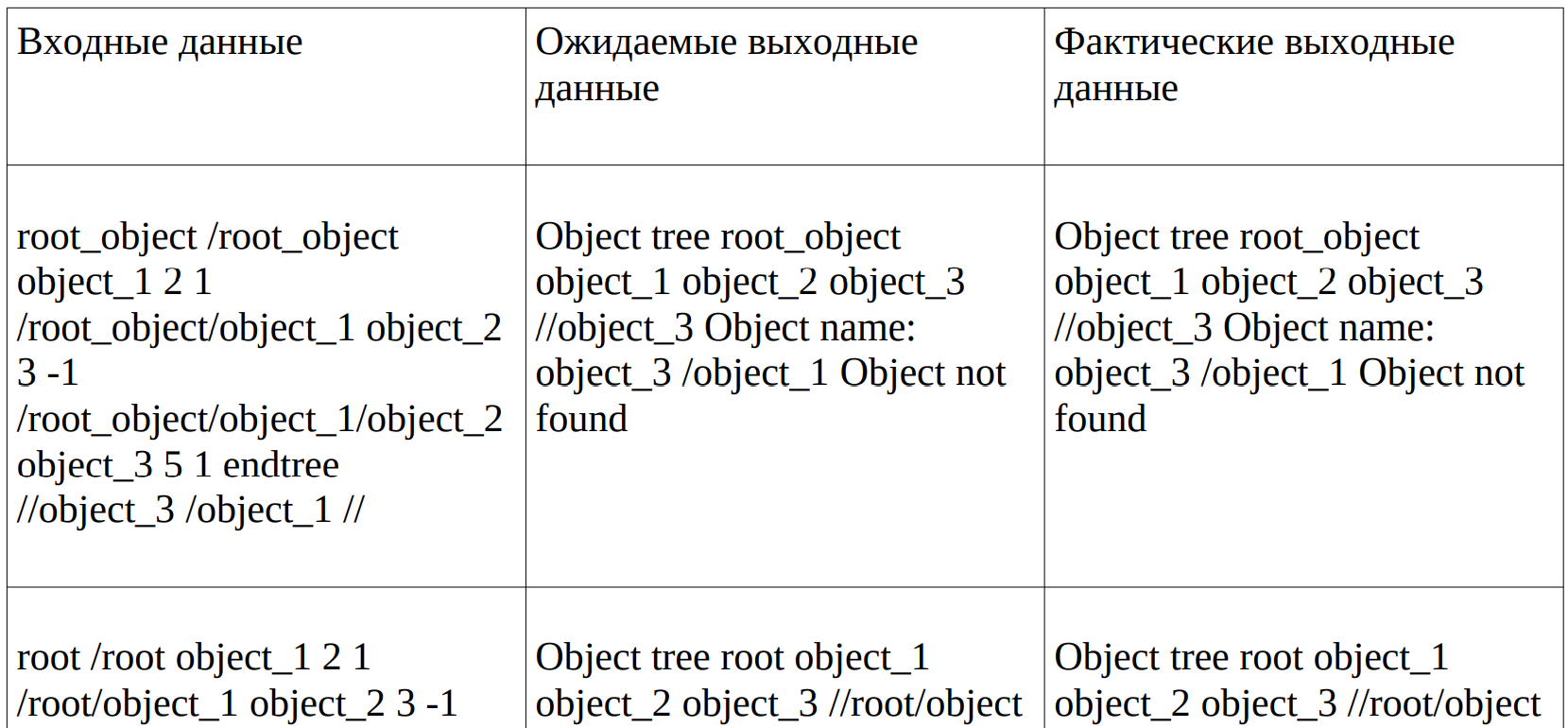
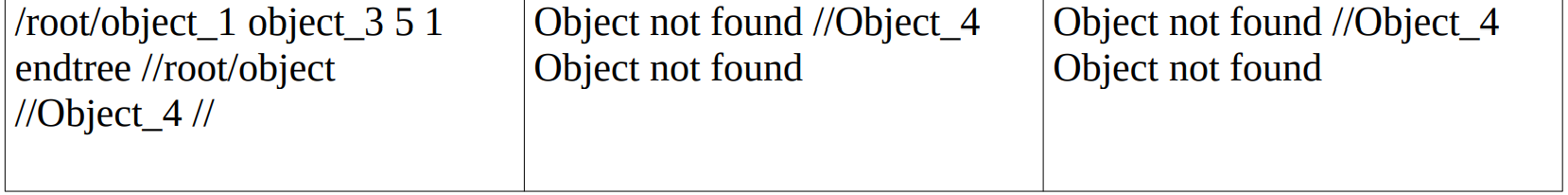
app.build\_tree();

return (app.exec\_app());

}

Тестирование

Таблица 13 «результаты тестирования программы»



Заключение

В ходе работы научился:

* Разработке базового класса для объектов;
* Определению общего функционала для используемых в рамках приложения объектов;
* Построению дерева иерархии объектов;
* Алгоритмам обработки структур данных в виде дерева;
* Поиску объекта на дереве иерархии по координатам;
* Выводу дерева иерархии объектов;
* Поиску указателя на объект по координате на дереве иерархии объектов;
* Логическому анализу данных;
* Реализации алгоритма решения задачи посредством поиска объектов и указателей на них на дереве иерархии объектов

Список Используемой Литературы

1. Васильев А.Н. Объектно-ориентированное программирование на С++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.
2. Шилдт Г. С++: 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://onlineedu.mirea.ru/mod/resource/view.php?id=247030 (дата обращения 05.05.2021).
4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».
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).