Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

```
«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»
```

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Поиск головного объекта выполняется от последнего созданного объекта. Первоначально последним созданным объектом считается корневой объект. Если для головного объекта обнаруживается дубляж имени в непосредственно подчиненных объектах, то объект не создается. Если обнаруживается дубляж имени на дереве иерархии объектов, то объект не создается. Если номер класса объекта задан некорректно, то объект не создается.

Вывод иерархического дерева объектов на консоль.

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:

- метод поиска объекта на ветке дереве иерархии от текущего по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на искомой ветке дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод вывода иерархии объектов (дерева или ветки) от текущего объекта (допускается использовать один целочисленный параметр со значением по-умолчанию);
- метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта (допускается использовать один целочисленный параметр со значением по-умолчанию);
- метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Устаревший метод вывода из задачи КВ_1 убрать.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется. При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

1. Построить дерево объектов системы (в методе корневого объекта построения исходного дерева объектов).

2. В методе корневого объекта запуска моделируемой системы реализовать:2.1. Вывод на консоль иерархического дерева объектов в следующем виде:

```
root
    ob_1
        ob_2
    ob_3
        ob_4
            ob_5
        ob_6
            ob_7
```

где: root - наименование корневого объекта (приложения).

2.2. Переключение готовности объектов согласно входным данным (командам). 2.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

```
root  is ready
    ob_1  is ready
        ob_2  is ready
    ob_3  is ready
        ob_4 is not ready
            ob_5 is not ready
        ob_6 is ready
            ob_7 is not ready
```

1.1 Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Последовательность ввода организовано так, что головной объект для очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка:

```
«Наименование корневого объекта»
```

Со второй строки:

```
«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»
. . . . .
endtree
```

Со следующей строки вводятся команды включения или отключения объектов

```
«Наименование объекта» «Номер состояния объекта»
```

Пример ввода:

```
app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
```

1.2 Описание выходных данных

Вывести иерархию объектов в следующем виде:

```
Object tree
«Наименование корневого объекта»
    «Наименование объекта 1»
        «Наименование объекта 2»
    «Наименование объекта 3»
. . . . .
The tree of objects and their readiness
«Наименование корневого объекта» «Отметка готовности»
    «Наименование объекта 1» «Отметка готовности»
        «Наименование объекта 2» «Отметка готовности»
    «Наименование объекта 3» «Отметка готовности»
. . . . .
«Отметка готовности» - равно «is ready» или «is not ready»
```

Отступ каждого уровня иерархии 4 позиции.

Пример вывода:

```
Object tree
app_root
    object_01
        object_07
```

```
    object_02
        object_04
        object_05
The tree of objects and their readiness
app_root is ready
    object_01 is ready
        object_07 is not ready
    object_02 is ready
        object_04 is ready
        object_05 is not ready
```

2 МЕТОД РЕШЕНИЯ

Класс base_class:

- свойства/поля:
    - поле готовность объекта:
        - наименование — status;
        - тип — int;
        - модификатор доступа — private;
- функционал:
    - метод search_in_branch — метод поиска на ветке по имени;
    - метод search_in_tree — метод поиска на дереве по имени;
    - метод print_branch_status — метод вывода иерархии объектов и отметок их готовности (дерева или ветки) от текущего объекта;
    - метод print_branch — метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
    - метод set_status — метод установки готовности объекта.

Класс application:

- функционал:
    - метод build_tree — строит дерево иерархии объектов;
    - метод exec_app — запуск системы.

Класс cl2:

- функционал:
    - метод cl2 — параметризированный конструктор.

Класс cl3:

- функционал:
    - метод cl3 — параметризированный конструктор.

Класс cl4:

- функционал:
    - метод cl4 — параметризированный конструктор.

Класс cl5:

- функционал:
    - метод cl5 — параметризированный конструктор.

Класс cl6:

- функционал:
    - метод cl6 — параметризированный конструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	base_class				
		application	public		2
		cl2	public		3
		cl3	public		4
		cl4	public		5
		cl5	public		6
		cl6	public		7
2	application				
3	cl2			Второй производный класс	
4	cl3			Третий производный класс	
5	cl4			Четвертый производный класс	
6	cl5			Пятый производный класс	
7	cl6			Шестой производный класс	

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода search_in_branch класса base_class

Функционал: метод поиска на ветке по имени.

Параметры: string name - имя объекта для поиска.

Возвращаемое значение: base_class* - указатель на найденный объект.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода search_in_branch класса base_class

№	Предикат	Действия	№ перехода
1	результат метода get_name равен значению параметра name?	возврат this	∅
			2
2	проход по всем подчиненным объекта	инициализация переменной object значением результата вызова метода search_in_branch(name) объекта child	3
		возврат nullptr	∅
3	object существует?	возврат object	∅
			2

3.2 Алгоритм метода set_status класса base_class

Функционал: метод установки готовности объекта.

Параметры: int status - параметр готовности объекта.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода set_status класса base_class

№	Предикат	Действия	№ перехода
1	значение параметра status равно 0?	присваивание полю объекта status значение 0	2
		инициализация объекта parent класса base_class* значением результата метода get_parent()	3
2	проход по всем подчиненным объекта	вызов метода set_status(0) объекта child	2
			∅
3	parent существует?		4
			8
4	поле status указателя parent равно 0?		5
		присваивание переменной parent значение результата метода get_parent() объекта parent	7
5		присваивание полю объекта status значение 0	6
6		возврат	∅
7		присваивание объекту parent значение результата метода get_parent() объекта parent	3
8		присваивание полю status значение аргумента status	∅

3.3 Алгоритм метода print_branch_status класса base_class

Функционал: метод вывода иерархии объектов и отметок их готовности (дерева или ветки) от текущего объекта.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода print_branch_status класса base_class

№	Предикат	Действия	№ перехода
1		инициализация объекта parent класса base_class* значением результата метода get_parent()	2
2	parent существует?	вывод на экран " " (4 пробела)	3
		вывод на экран name и " is "	4
3		присваивание parent результата метода get_parent() объекта parent	2
4	значение поля status равно 0?	вывод на экран "not "	5
			5
5		вывод на экран "ready\n"	6
6	проход по всем подчиненным объекта	вызов метода print_branch_status() объекта child	6
			∅

3.4 Алгоритм метода print_branch класса base_class

Функционал: метод вывода иерархии объектов (дерева или ветки) от текущего объекта.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода print_branch класса base_class

№	Предикат	Действия	№ перехода
1		инициализация объекта parent класса base_class* значением результата метода get_parent()	2

№	Предикат	Действия	№ перехода
2	parent существует?	вывод на экран " " (4 пробела)	3
		вывод на экран name и переход на новую строку	4
3		присваивание parent результата метода get_parent() объекта parent	2
4	проход по всем подчиненным объекта	вызов метода print_branch() объекта child	4
			∅

3.5 Алгоритм конструктора класса cl2

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в качестве параметров указателя на головной объект и имени на текущий.

Параметры: base_class* parent - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 6.

Таблица 6 – Алгоритм конструктора класса cl2

№	Предикат	Действия	№ перехода
1			∅

3.6 Алгоритм конструктора класса cl3

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в качестве параметров указателя на головной объект и имени на текущий.

Параметры: base_class* parent - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 7.

Таблица 7 – Алгоритм конструктора класса cl3

№	Предикат	Действия	№ перехода
1			∅

3.7 Алгоритм конструктора класса cl4

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в качестве параметров указателя на головной объект и имени на текущий.

Параметры: base_class* parent - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl4

№	Предикат	Действия	№ перехода
1			∅

3.8 Алгоритм конструктора класса cl5

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в качестве параметров указателя на головной объект и имени на текущий.

Параметры: base_class* parent - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl5

№	Предикат	Действия	№ перехода
1			∅

3.9 Алгоритм конструктора класса cl6

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в качестве параметров указателя на головной объект и имени на текущий.

Параметры: base_class* parent - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl6

№	Предикат	Действия	№ перехода
1			∅

3.10 Алгоритм метода build_tree класса application

Функционал: строит дерево иерархии объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода build_tree класса application

№	Предикат	Действия	№ перехода
1		объявление переменных number типа int, parent_name и child_name типа string	2
2		ввод значения parent_name с клавиатуры	3

№	Предикат	Действия	№ перехода
3		вызов метода set_name() с параметром parent_name	4
4		инициализация объекта parent класса base_class* указателем на данный объект	5
5	ввод значения переменной parent_name с клавиатуры И её значение не равно строке "endtree"	ввод значений child_name и number	6
			∅
6	parent не существует ИЛИ имя parent не равно значению переменной parent_name	присваивание объекту parent результата метода search_in_tree() с параметром parent_name	7
			7
7	parent существует И у parent нет подчиненного объекта с именем значения переменной child_name И в дереве нет объекта с именем значения переменной child_name		8
			5
8	значение переменной number равно 2?	создание нового объекта производного класса cl2	5
	значение переменной number равно 3?	создание нового объекта производного класса cl3	5
	значение переменной number равно 4?	создание нового объекта производного класса cl4	5
	значение переменной number равно 5?	создание нового объекта производного класса cl5	5
	значение переменной number	создание нового объекта производного класса cl6	5

№	Предикат	Действия	№ перехода
	равно 6?		
			5

3.11 Алгоритм метода exec_app класса application

Функционал: запуск системы.

Параметры: нет.

Возвращаемое значение: int - код ощибки.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		вывод сообщения "Object tree\n"	2
2		вызов метода print_branch()	3
3		объявление переменных status типа int и name типа string	4
4	ввод значений переменных name и status	инициализация объекта object класса base_class* значением результата метода search_in_tree() с параметром name	5
			6
5	object существует?	вызов метода set_status() с параметром status объекта object	4
			4
6		вывод на экран перехода на новую строку, сообщения "The tree of objects and their readiness\n"	7
7		вызов метода print_branch_state()	8
8		возврат 0	∅

3.12 Алгоритм функции main

Функционал: основная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 13.

Таблица 13 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта tree класса application с параметром nullptr	2
2		вызов метода build_tree() объекта tree	3
3		возврат результата вызова метода exec_app() объекта tree	∅

3.13 Алгоритм метода search_in_tree класса base_class

Функционал: метод поиска на дереве по имени.

Параметры: string name - имя объекта для поиска.

Возвращаемое значение: base_class* - указатель на найденный объект.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода search_in_tree класса base_class

№	Предикат	Действия	№ перехода
1		иниицализация объекта parent класса base_class* значением указателя на данный объект	2
2	результат метода get_parent от указателя parent не равен nullptr?	присваивание parent значение результата метода get_parent() объекта parent	2
		возврат результата метода search_in_branch() с праметром name объекта parent	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-9.

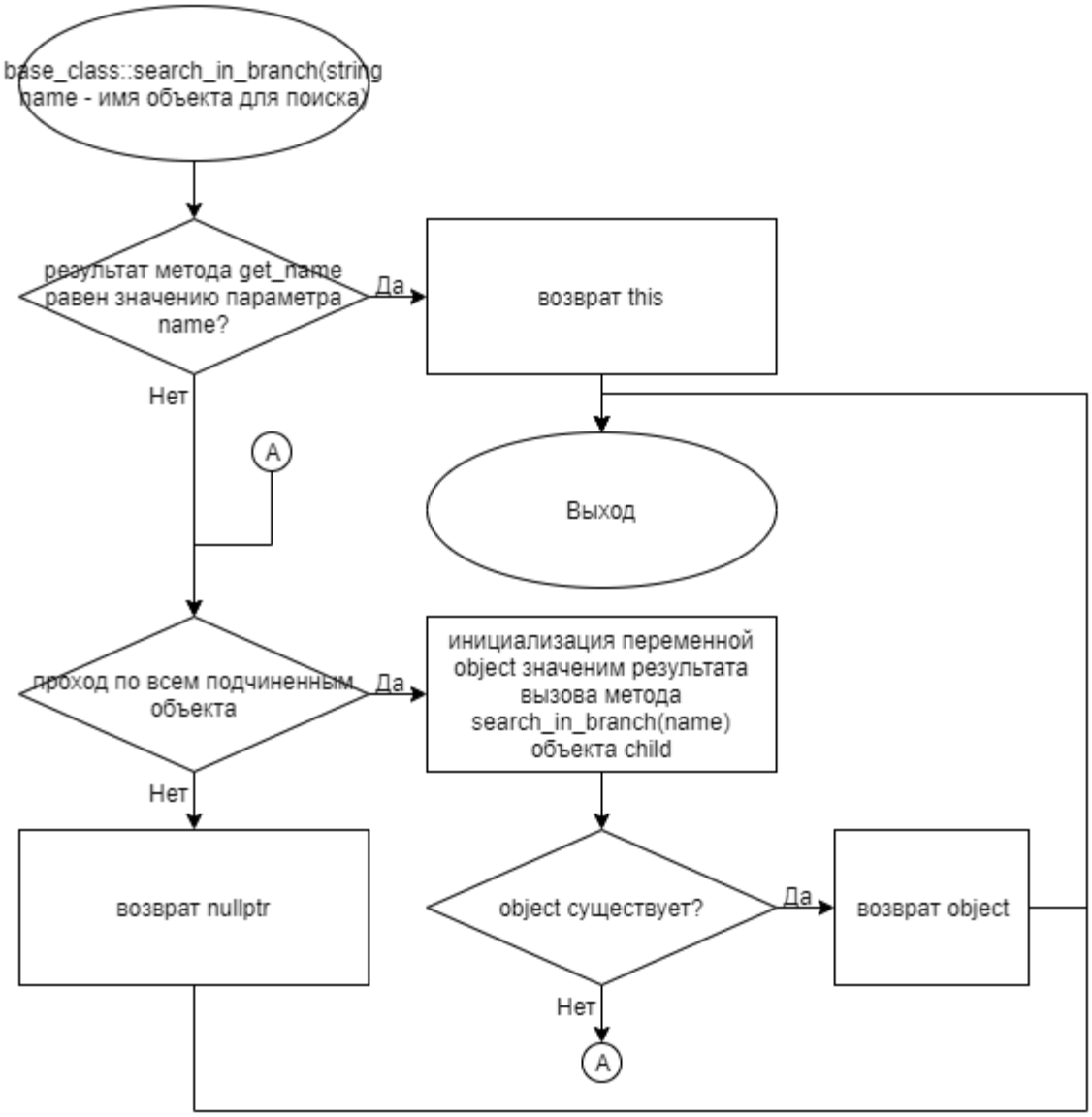

Рисунок 1 – Блок-схема алгоритма

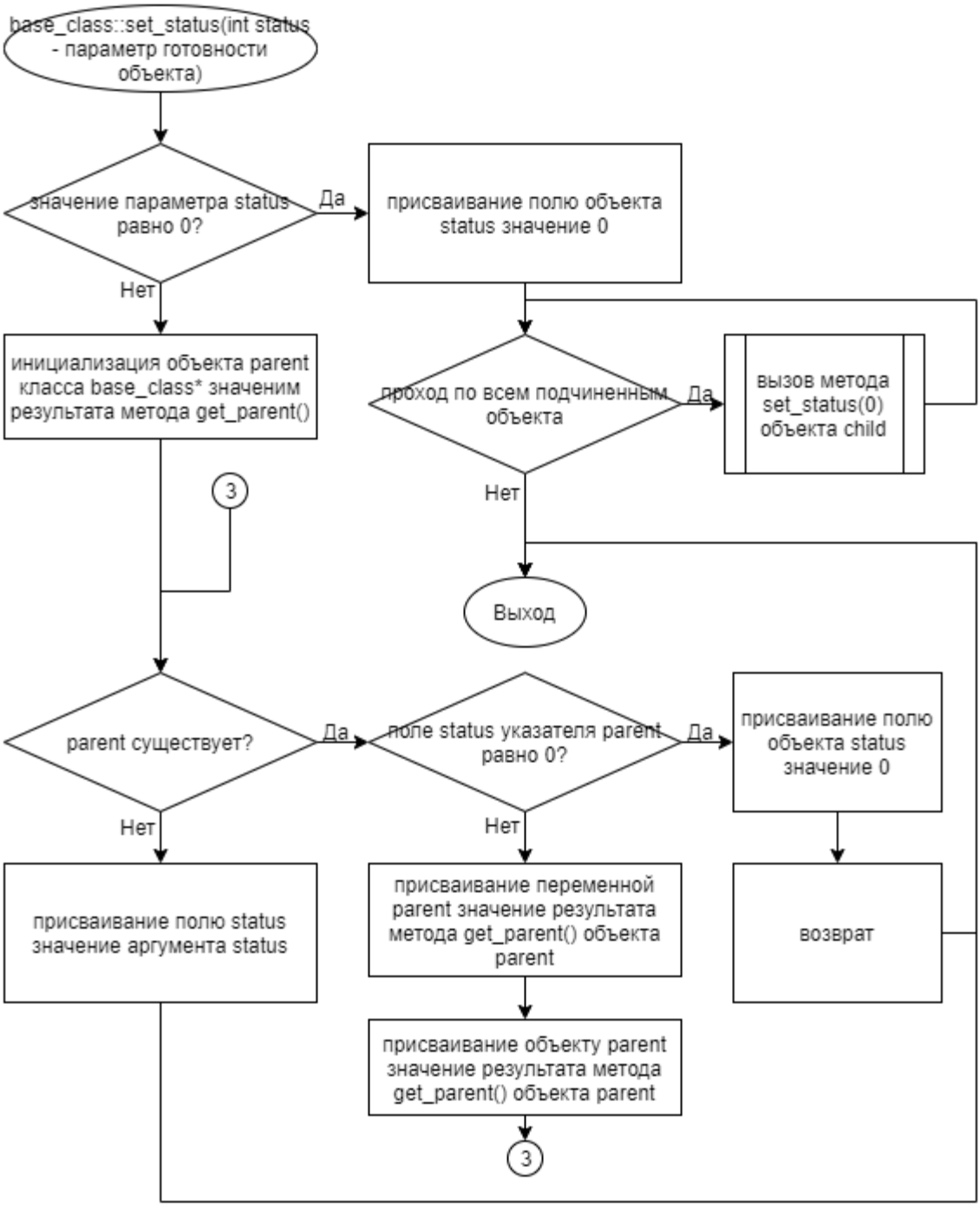

Рисунок 2 – Блок-схема алгоритма

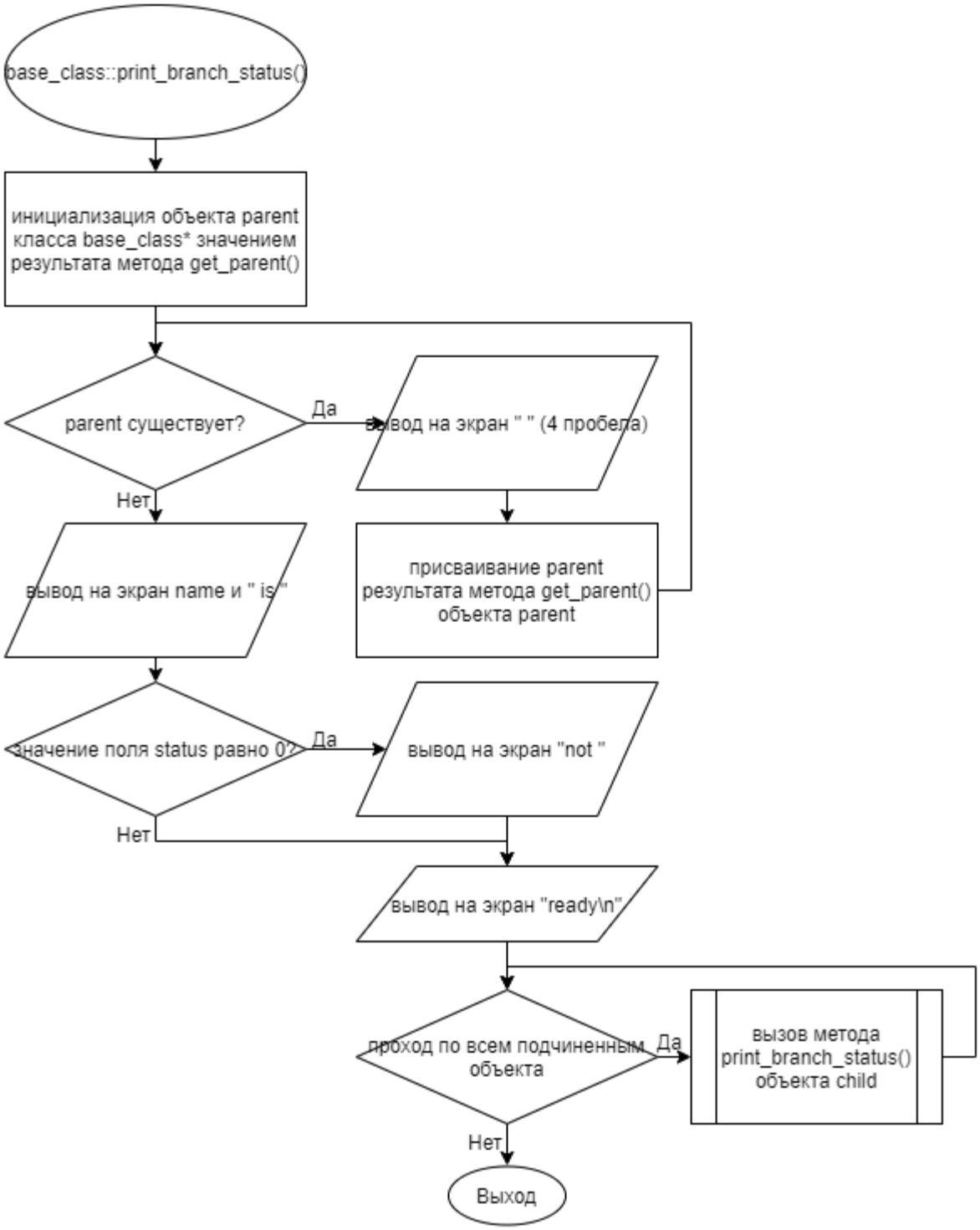

Рисунок 3 – Блок-схема алгоритма

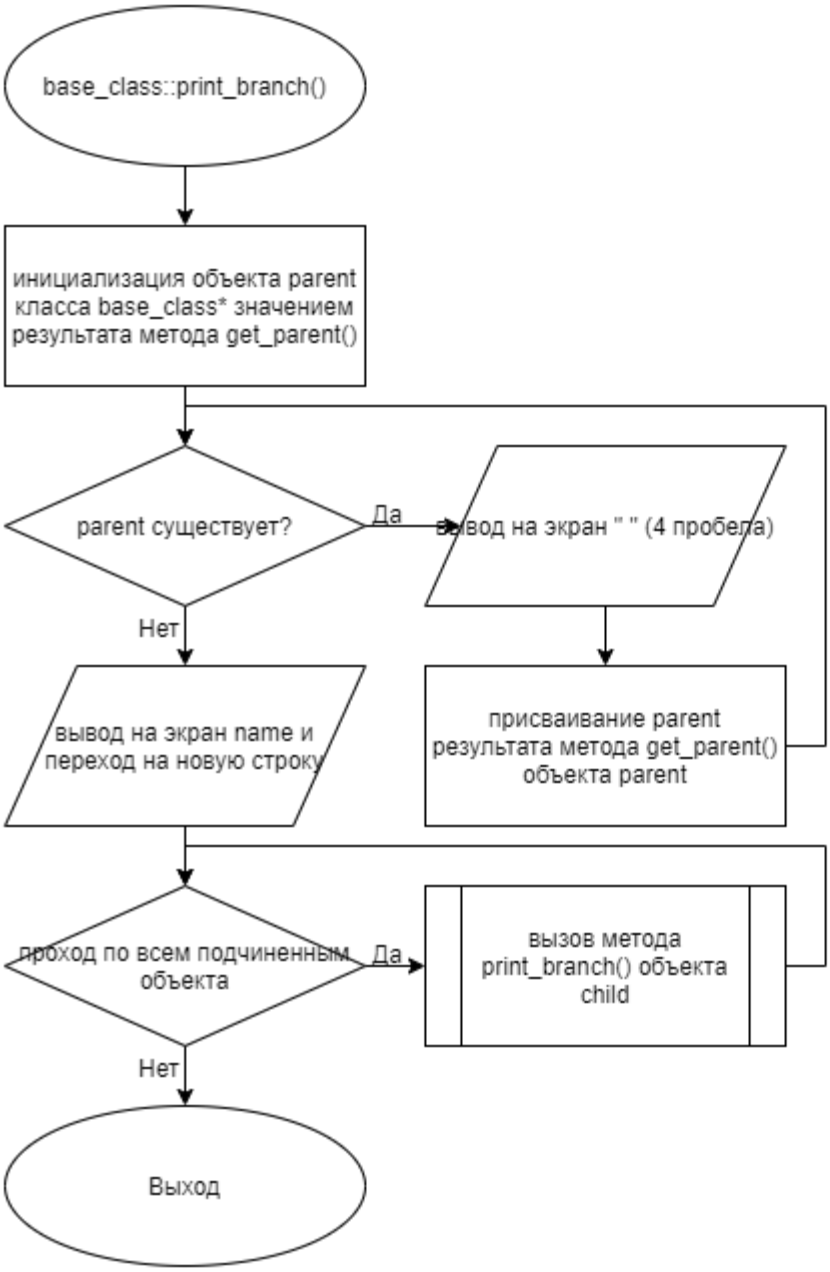

Рисунок 4 – Блок-схема алгоритма

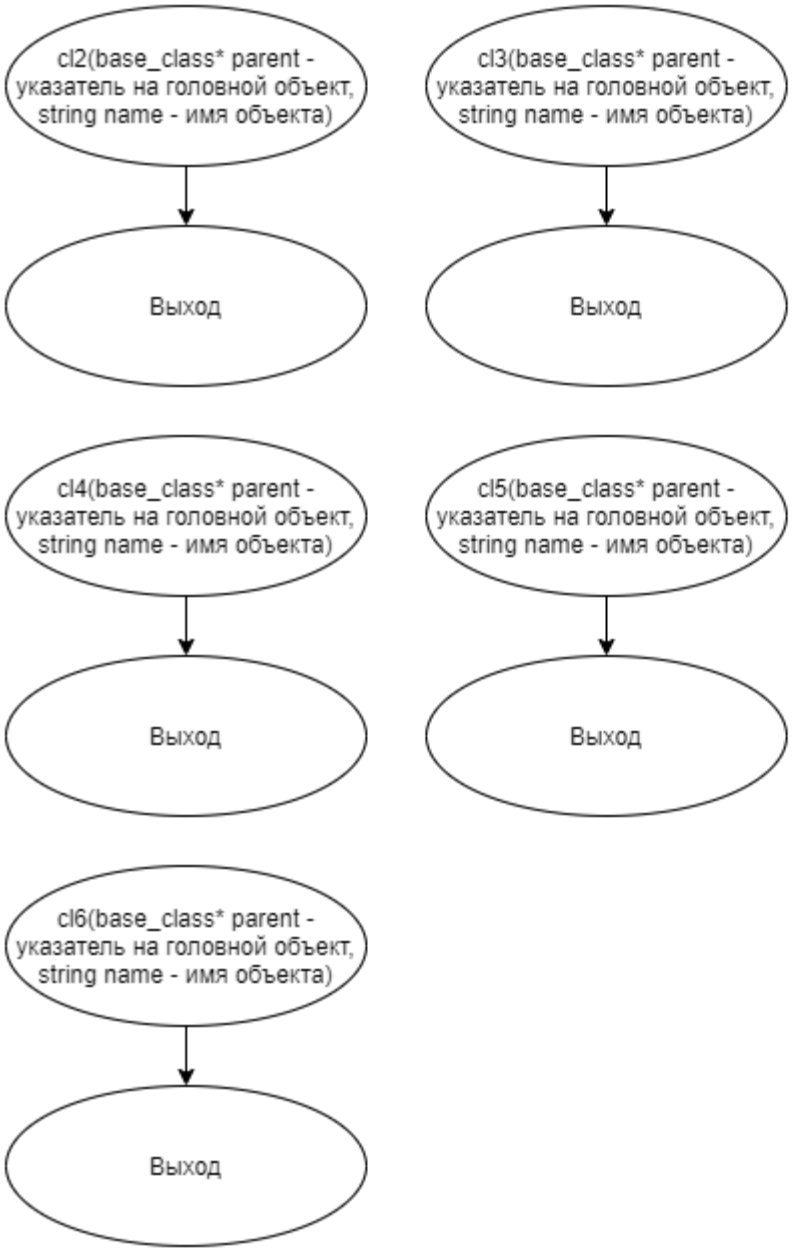

Рисунок 5 – Блок-схема алгоритма

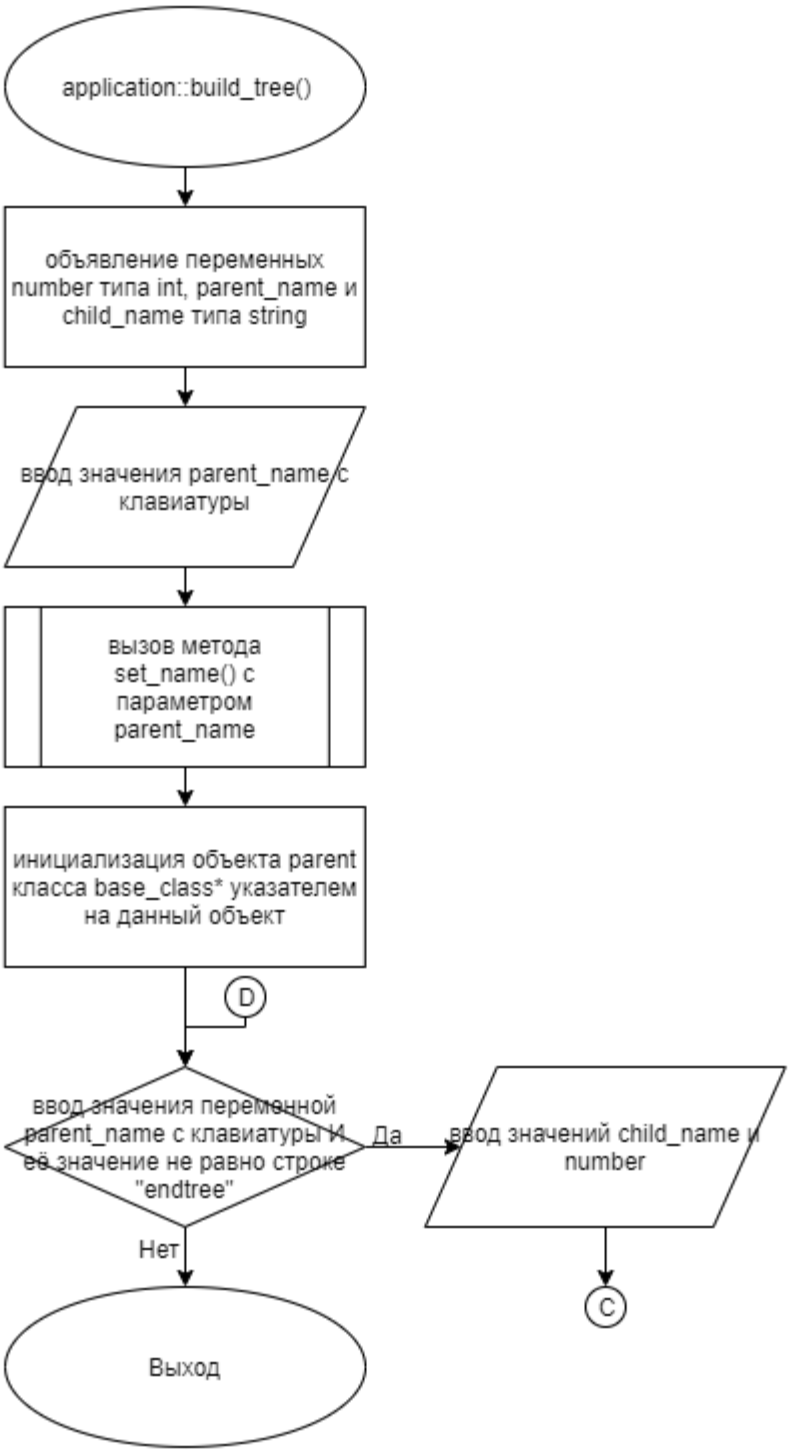

Рисунок 6 – Блок-схема алгоритма

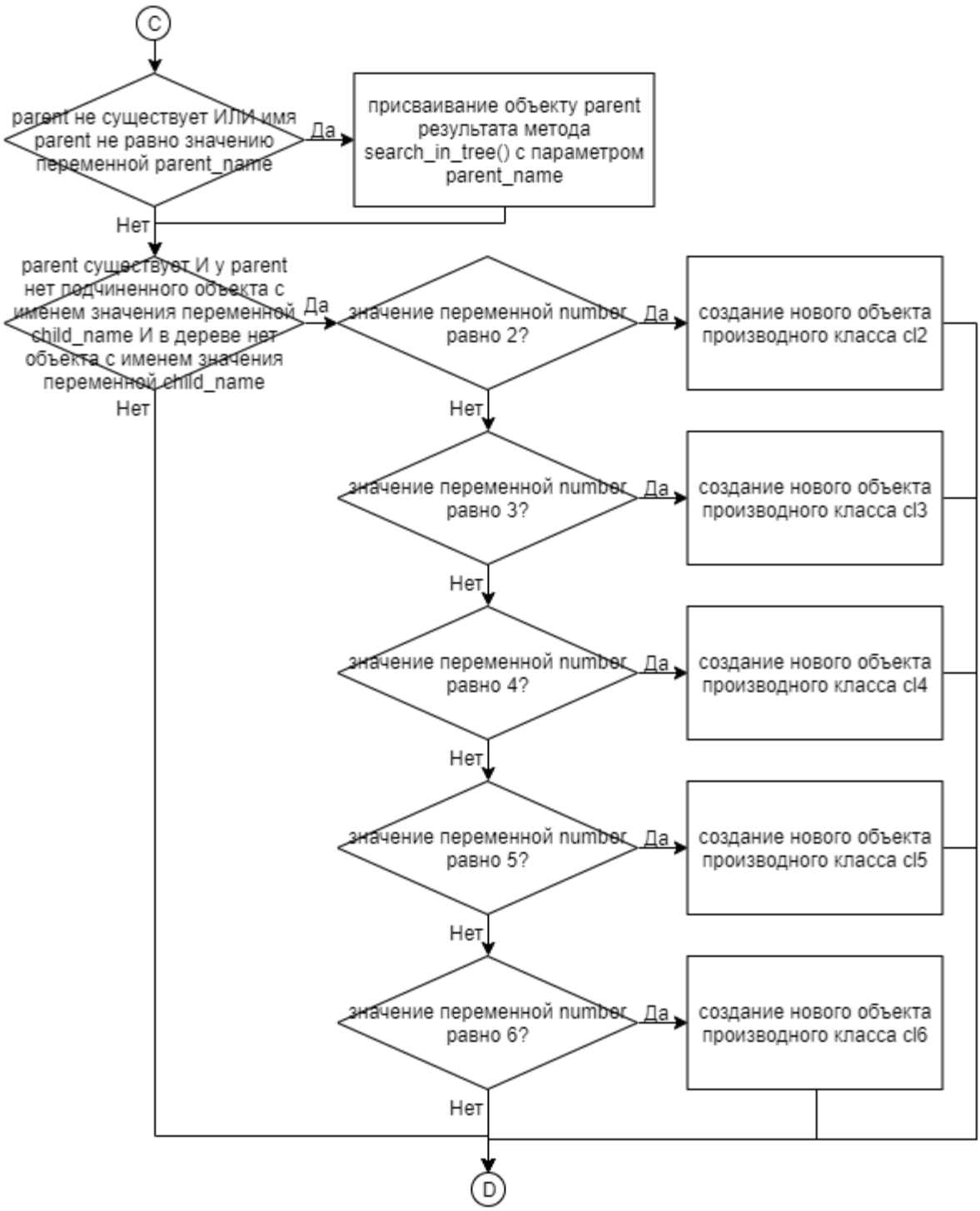

Рисунок 7 – Блок-схема алгоритма

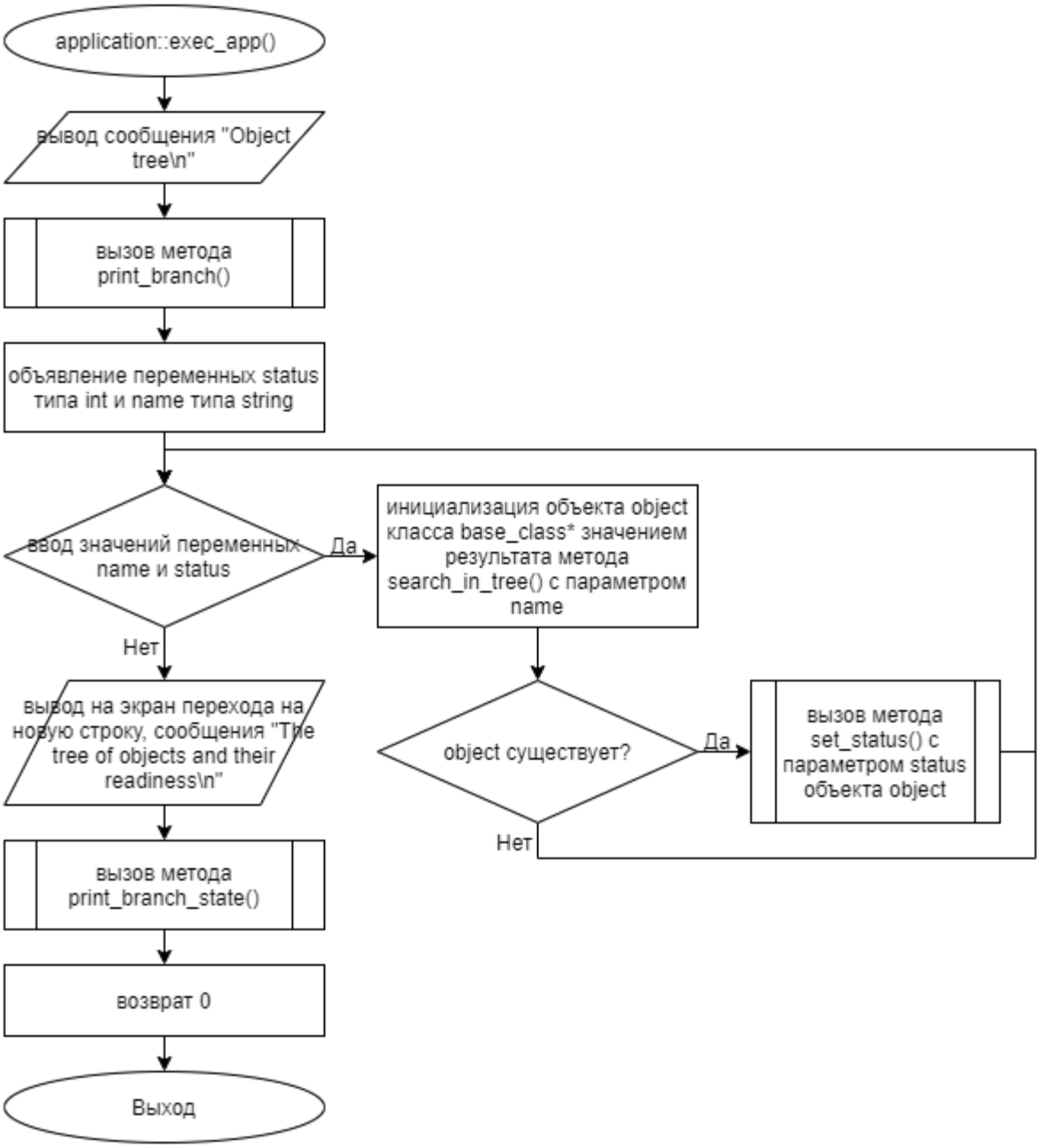

Рисунок 8 – Блок-схема алгоритма

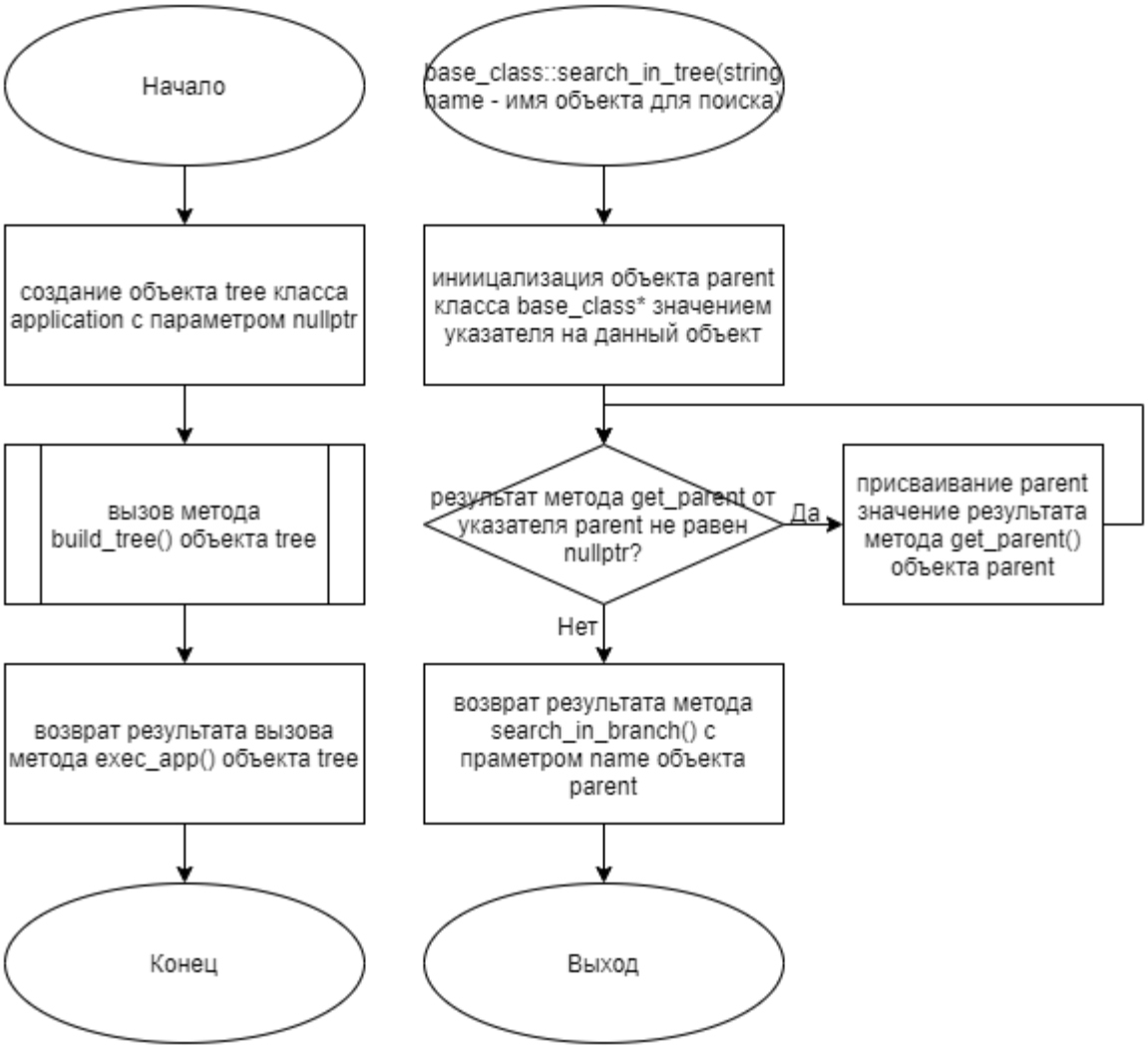

Рисунок 9 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
#include "cl2.h"
#include "cl3.h"
#include "cl4.h"
#include "cl5.h"
#include "cl6.h"
#include <iostream>

application::application(base_class*      root,      std::string      name):
base_class(root, name)
{ }
void application::build_tree()
{
   int number; std::string parent_name, child_name;
   std::cin >> parent_name;
   set_name(parent_name);
   base_class* parent = this;
   while (std::cin >> parent_name && parent_name != "endtree")
   {
      std::cin >> child_name >> number;
      if (!parent || parent -> get_name() != parent_name)
         parent = search_in_tree(parent_name);
      if  (parent  &&  !parent  ->  get_child(child_name)  &&  !parent  ->
search_in_tree(child_name))
      {
         switch (number)
         {
            case 2:
               new cl2(parent, child_name);
               break;
            case 3:
               new cl3(parent, child_name);
               break;
            case 4:
               new cl4(parent, child_name);
               break;
            case 5:
               new cl5(parent, child_name);
               break;
```

```
            case 6:
               new cl6(parent, child_name);
               break;
            default:
               break;
         }
      }
   }
}
int application::exec_app()
{
   std::cout << "Object tree\n";
   print_branch();
   int status; std::string name;
   while (std::cin >> name >> status)
   {
      base_class* object = search_in_tree(name);
      if (object) object -> set_status(status);
   }
   std::cout << "The tree of objects and their readiness\n";
   print_branch_status();
   return 0;
}
```

5.2 Файл application.h

Листинг 2 – application.h

```
#ifndef __APPLICATION__H
#define __APPLICATION__H
#include "base_class.h"

class application: public base_class
{
public:
   application(base_class* root, std::string name = "root");
   void build_tree(); //changed
   int exec_app();
};

#endif
```

5.3 Файл base_class.cpp

Листинг 3 – base_class.cpp

```
#include "base_class.h"
#include <iostream>

base_class::base_class(base_class* parent, std::string name)
{
   this -> parent = parent;
   this -> name = name;
   if (parent) parent -> children.push_back(this);
}
base_class::~base_class()
{ for (base_class* child: children) delete child; }

bool base_class::set_name(std::string name)
{
   if (parent && parent -> get_child(name))
      return false;
   this -> name = name;
   return true;
}
std::string base_class::get_name()
{ return name; }
base_class* base_class::get_parent()
{ return parent; }
base_class* base_class::get_child(std::string name)
{
   for (base_class* child: children)
      if (child -> get_name() == name)
         return child;
   return nullptr;
}

void base_class::print_branch()
{
   base_class* parent = get_parent();
   while (parent)
   {
      std::cout << "    ";
      parent = parent -> get_parent();
   }
   std::cout << name << std::endl;
   for (base_class* child: children)
      child -> print_branch();
}
void base_class::print_branch_status()
{
   base_class* parent = get_parent();
   while (parent)
   {
      std::cout << "    ";
      parent = parent -> get_parent();
```

```
   }
   std::cout << name << " is ";
   if (!status)
      std::cout << "not ";
   std::cout << "ready\n";
   for (base_class* child: children)
      child -> print_branch_status();
}

base_class* base_class::search_in_branch(std::string name)
{
   if (get_name() == name) return this;
   for (base_class* child: children)
   {
      base_class* object = child -> search_in_branch(name);
      if (object) return object;
   }
   return nullptr;
}
base_class* base_class::search_in_tree(std::string name)
{
   base_class* parent = this;
   while (parent -> get_parent())
      parent = parent -> get_parent();
   return parent -> search_in_branch(name);
}

void base_class::set_status(int status)
{
   if (status == 0)
   {
      this -> status = 0;
      for (base_class* child: children)
         child -> set_status(0);
   }
   else
   {
      base_class* parent = get_parent();
      while (parent)
      {
         if (parent -> status == 0)
         {
            this -> status = 0;
            return;
         }
         parent = parent -> get_parent();
      }
      this -> status = status;
   }
}
```

5.4 Файл base_class.h

Листинг 4 – base_class.h

```
#ifndef __BASE_CLASS__H
#define __BASE_CLASS__H

#include <string>
#include <vector>

class base_class
{
private:
   std::string name;
   base_class* parent;
   std::vector <base_class*> children;
   int status = 0; //new
public:
   base_class(base_class* parent, std::string name = "default");
   ~base_class();
   bool set_name(std::string name);
   std::string get_name();
   base_class* get_parent();
   base_class* get_child(std::string name);
   void print_branch(); //changed
   void print_branch_status(); //new
   base_class* search_in_branch(std::string name); //new
   base_class* search_in_tree(std::string name); //new
   void set_status(int status); //new
};
#endif
```

5.5 Файл cl2.cpp

Листинг 5 – cl2.cpp

```
#include "cl2.h"

cl2::cl2(base_class* parent, std::string name): base_class(parent, name)
{ }
```

5.6 Файл cl2.h

Листинг 6 – cl2.h

```
#ifndef __CL2__H
#define __CL2__H
#include "base_class.h"

class cl2: public base_class
{
public:
   cl2(base_class* root, std::string name = "cl2");
};

#endif
```

5.7 Файл cl3.cpp

Листинг 7 – cl3.cpp

```
#include "cl3.h"

cl3::cl3(base_class* parent, std::string name): base_class(parent, name)
{ }
```

5.8 Файл cl3.h

Листинг 8 – cl3.h

```
#ifndef __CL3__H
#define __CL3__H
#include "base_class.h"

class cl3: public base_class
{
public:
   cl3(base_class* root, std::string name = "cl3");
};

#endif
```

5.9 Файл cl4.cpp

Листинг 9 – cl4.cpp

```
#include "cl4.h"

cl4::cl4(base_class* parent, std::string name): base_class(parent, name)
{ }
```

5.10 Файл cl4.h

Листинг 10 – cl4.h

```
#ifndef __CL4__H
#define __CL4__H
#include "base_class.h"

class cl4: public base_class
{
public:
   cl4(base_class* root, std::string name = "cl4");
};

#endif
```

5.11 Файл cl5.cpp

Листинг 11 – cl5.cpp

```
#include "cl5.h"

cl5::cl5(base_class* parent, std::string name): base_class(parent, name)
{ }
```

5.12 Файл cl5.h

Листинг 12 – cl5.h

```
#ifndef __CL5__H
#define __CL5__H
#include "base_class.h"

class cl5: public base_class
{
public:
   cl5(base_class* root, std::string name = "cl5");
};

#endif
```

5.13 Файл cl6.cpp

Листинг 13 – cl6.cpp

```
#include "cl6.h"

cl6::cl6(base_class* parent, std::string name): base_class(parent, name)
{ }
```

5.14 Файл cl6.h

Листинг 14 – cl6.h

```
#ifndef __CL6__H
#define __CL6__H
#include "base_class.h"

class cl6: public base_class
{
public:
   cl6(base_class* root, std::string name = "cl6");
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "application.h"

int main()
{
   application tree(nullptr);
   tree.build_tree();
   return tree.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 15.

Таблица 15 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
tree_root tree_root object_1 3 tree_root object_2 2 object_2 object_4 3 object_2 object_5 5 object_1 object_6 2 endtree tree_root 1 object_6 3 object_1 1 object_2 -2 object_4 1	Object tree tree_root object_1 object_6 object_2 object_4 object_5 The tree of objects and their readiness tree_root is ready object_1 is ready object_6 is not ready object_2 is ready object_4 is ready object_5 is not ready	Object tree tree_root object_1 object_6 object_2 object_4 object_5 The tree of objects and their readiness tree_root is ready object_1 is ready object_6 is not ready object_2 is ready object_4 is ready object_5 is not ready

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения 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).

3. 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).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

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).