Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

Расширить функциональность базового класса:

- метод переопределения головного объекта для текущего в дереве иерархии. Метод должен иметь один параметр, указатель на объект базового класса, содержащий указатель на новый головной объект. Переопределение головного объект для корневого объекта недопустимо. Недопустимо создать второй корневой объект. Недопустимо при переопределении, чтобы у нового головного появились два подчиненных объекта с одинаковым наименованием. Новый головной объект не должен принадлежать к объектам из ветки текущего. Если переопределение выполнено, метод возвращает значение «истина», иначе «ложь»;
- метод удаления подчиненного объекта по наименованию. Если объект не найден, то метод завершает работу. Один параметр строкового типа, содержит наименование удаляемого подчиненного объекта;
- метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задаться в следующем виде:
    - / - корневой объект;
    - //«имя объекта» - поиск объекта по уникальной имени от корневого (для однозначности уникальность требуется в рамках дерева);
    - . - текущий объект;
    - .«имя объекта» - поиск объекта по уникальной имени от текущего (для однозначности уникальность требуется в рамках ветви дерева от

текущего объекта);

- «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_2/ob_3
/ob_1/ob_2/ob_3
```

Если координата - пустая строка или объект не найден или определяется неоднозначно (дубляж имен на ветке, на дереве), тогда вернуть нулевой указатель.

Наименование объекта не содержит символы «.» и «/».

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 необходимо соблюдать. Если это требование исходя из входных данных нарушается, то соответствующий подчиненный объект не создается.

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). Если номер класса объекта задан некорректно, то объект не создается.

Собранная система отрабатывает следующие команды:

- SET «координата» – устанавливает текущий объект;
- FIND «координата» – находит объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» задает новый головной объект;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим. При вводе данных в названии команд ошибок нет. Если при переопределении головного объекта нарушается уникальность наименований подчиненных объектов для нового головного, переопределение не производится.

1.1 Описание входных данных

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.

После ввода состава дерева иерархии построчно вводятся команды:

- SET «координата» – установить текущий объект;
- FIND «координата» – найти объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» соответствует новому головному объекту;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершить функционирование системы (выполнение программы).

Команды SET, FIND, MOVE и DELETE вводятся произвольное число раз.

Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов:

```
rootela
/ 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_7
FIND object_4/object_7
FIND .
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева как в 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 с кодом возврата 1.

Если при построении при попытке создания объекта обнаружен дубляж, то вывести:

«координата головного объекта» Dubbing the names of subordinate objects

Если дерево построено, то далее построчно вводятся команды.

Для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

The object was not found at the specified coordinate: «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

Для команд MOVE вывести:

New head object: «наименование нового головного объекта»

Если головной объект не найден, то:

«искомая координата объекта» Head object is not found

Если переопределить головной объект не удалось, то:

«искомая координата объекта» Redefining the head object failed

Если у нового головного объекта уже есть подчиненный с таким же именем, то вывести:

«искомая координата объекта» Dubbing the names of subordinate objects

При попытке переподчинения головного объекта к объекту на ветке, вывести:

«координата нового головного объекта» Redefining the head object failed

Для команды DELETE:

Если подчиненный объект удален, то вывести:

The object «абсолютный путь удаленного объекта» has been deleted

Если объект не найден, то ничего не выводить.

После команды END с новой строки вывести:

```
Current object hierarchy tree
```

Со следующей строки вывести текущую иерархию дерева.

Пример вывода иерархии дерева объектов:

```
Object tree
rootela
    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_7     Object is not found
object_4/object_7     Object name: object_7
.     Object name: object_2
.object_7     Object name: object_7
object_4/object_7     Object name: object_7
.object_7     Redefining the head object failed
Object is set: object_7
//object_1     Dubbing the names of subordinate objects
New head object: object_3
Current object hierarchy tree
rootela
    object_1
        object_7
    object_2
        object_4
        object_5
        object_3
    object_3
        object_7
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- функция count для лямбда-функция для подсчета количтсва объектво с заданным именем.

Класс application:

- функционал:
    - метод build_tree — создает дерево иерархии;
    - метод exec_app — запускает приложение.

Класс base_class:

- функционал:
    - метод find — метод нахождения объекта;
    - метод delete_child — метод удаления ребёнка;
    - метод set_parent — метод установки родителя;
    - метод search_in_branch — метод поиска объекта на ветви.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	application			класс "приложение"	
2	base_class			базовый класс программы	
		application	public		1

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 функции main

Функционал: основная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта tree класса application с помощью параметризированного конструктора с аргументом nullptr	2
2		вызов метода build_tree объекта tree	3
3		возврат значения результата метода exec_app объекта tree	∅

3.2 Алгоритм метода build_tree класса application

Функционал: создает дерево иерархии.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода build_tree класса application

№	Предикат	Действия	№ перехода
1		создание переменных number типа int, parent_name, child_name, parent_coords типа string	2
2		ввод значения parent_name с клавиатуры	3
3		вызов метода set_name с аргументом parent_name	4
4		инициализация указателя parent на объект класса base_class* указанием на данный объект	5
5	ввод значения переменной parent_coords и сравнение со строкой "endtree"	ввод значений для переменных child_name и number с клавиатуры	6
			∅
6		присваивание указателю parent значение результата метода find с аргументом parent_coords	7
7	parent не существует?	вывод на экран сообщения "Object tree\n"	8
			11
8		вызов метода print_branch	9
9		вывод на экран "The head object ", parent_coords, " is not found\n"	10
10		завершение работы программы	∅
11	результат вызова метода get_child с параметром child_name от указателя parent существует?	вывод на экран parent_coords, " Dubbing the names of subordinate objects\n"	5
			12
12	значение переменной number равно 2?	создание нового объекта класса cl2 с аргументами parent и child_name	5
	значение переменной number равно 3?	создание нового объекта класса cl3 с аргументами parent и child_name	5

№	Предикат	Действия	№ перехода
	значение переменной number равно 4?	создание нового объекта класса cl4 с аргументами parent и child_name	5
	значение переменной number равно 5?	создание нового объекта класса cl5 с аргументами parent и child_name	5
	значение переменной number равно 6?	создание нового объекта класса cl6 с аргументами parent и child_name	5
			5

3.3 Алгоритм метода exec_app класса application

Функционал: запускает приложение.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		вывод на экран "Object tree\n"	2
2		вызов метода print_branch	3
3		создание строковой переменной command	4
4		инициализация указателя current на объект класса base_class указанием на данный объект	5
5	ввод значения переменной command и сравнение со строкой "END"		8
		вывод на экран "Current object hierarchy tree\n"	6
6		вызов метода print_branch	7
7		возврат значения 0	∅
8	значение переменной	создание строковой переменной name	9

№	Предикат	Действия	№ перехода
	command равно строке "DELETE"?		
	значение переменной command равно строке "SET" ИЛИ значение переменной command равно строке "FIND" ИЛИ значение переменной command равно строке "MOVE"?	создание строковой переменной coords	16
			5
9		ввод значения переменной name с клавиатуры	10
10		инициализация указателя child на объект класса base_class значением результата метода get_child указателя current с аргументом name	11
11	child существует?	инициализация строковой переменной path значением '/' + результат метода get_name указателя child	12
			5
12	результат метода get_parent указателя child существует?	присваивание переменной child значение результата метода get_parent указателя child	13
			14
13		присваивание переменной path значение '/' + child->get_name() + path	12
14		вызов метода delete_child указателя current с аргументом name	15
15		вывод на экран "The object ", path, " has been deleted\n"	5
16		создание строковой переменной coords	17
17		ввод значения переменной coords с клавиатуры	18

№	Предикат	Действия	№ перехода
18		инициализация указателя object на объект класса base_class значением результата вызова метода find указателя current с аргументом coords	19
19	значение переменной command равно строке "SET"?		20
	значение переменной command равно строке "FIND"?	вывод на экран coordinates, " Object "	23
	значение переменной command равно строке "MOVE"?		25
			5
20	object существует?	присваивание переменной current значение указателя object	21
		вывод на экран "The object was not found at the specified coordinate: ", coords	22
21		вывод на экран "Object is set: ", резуьтат метода get_name указателя current	22
22		вывод на экран переход на новую строку	5
23	object существует?	вывод на экран "name: ", результат метода get_name указателя object	24
		вывод на экран "is not found"	24
24		вывод на экран переход на новую строку	5
25	object не существует?	вывод на экран coordinates, " Head object is not found"	26
	результат метода get_child указателя object с параметром результата метода get_name указателя	вывод на экран coordinates, " Dubbing the names of subordinate objects"	26

№	Предикат	Действия	№ перехода
	current существует?		
	результат метода set_parent указателя current с параметром object равен false?	вывод на экран coordinates, " Redefining the head object failed"	26
		вывод на экран "New head object: ", результат метода get_name указателя object	26
26		вывод на экран переход на новую строку	5

3.4 Алгоритм метода delete_child класса base_class

Функционал: метод удаления ребёнка.

Параметры: string name - имя объекта для удаления.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода delete_child класса base_class

№	Предикат	Действия	№ перехода
1		инициализация указателя child типа base_class* результатом метода get_child с аргументом name	2
2	child существует?	инициализация целочисленной переменной i значением 0	3
			∅
3	значение переменной i меньше размера списка children?		4
			∅
4	элемент списка children с индексом i равен значению	из поля children вырезать объект с индексом childre.begin()+i	5

№	Предикат	Действия	№ перехода
	указателя child?		
		i++	3
5		удалить указатель child	6
6		завершить цикл for	∅

3.5 Алгоритм метода set_parent класса base_class

Функционал: метод установки родителя.

Параметры: base_class* object - указатель на новый головной объект.

Возвращаемое значение: bool - результат выполнению метода.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода set_parent класса base_class

№	Предикат	Действия	№ перехода
1	!object || !this->parent || object->get_child(this->get_name())	возврат false	∅
		инициализация указателя parent типа base_class* указателем object	2
2	parent		3
			4
3	parent == this	возврат false	∅
		присваивание указателю parent указание результат вызова метода get_parent от указателя parent	2
4		инициализация указателя current типа base_class* на поле parent данного объекта	5
5		инициализация целочисленной переменной i значением 0	6
6	i < current->children.size()		7
			8

№	Предикат	Действия	№ перехода
7	current->children[i] == this	из поля children данного объекта вырезать ребёнка с индексом current->children.begin()+i и завершить цикл for	8
		увеличить значение переменной i на 1	6
8		присвоить полю parent значение параметра object	9
9		указателю object добавить в конец списка children данный объект	10
10		возврат true	∅

3.6 Алгоритм метода search_in_branch класса base_class

Функционал: метод поиска объекта на ветви.

Параметры: string name - имя объекта для поиска.

Возвращаемое значение: base_class* - указатель на найденный объект.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода search_in_branch класса base_class

№	Предикат	Действия	№ перехода
1		инициализация лямбда-функции count для подсчёта количества объектов с именем name	2
2	значение результата выполнения лямбда-функции count с параметрами: значение переменной name и указатель this; не равно 1?	возврат nullptr	∅
			3
3	результат метода get_name равен значению переменной name?	возврат this	∅

№	Предикат	Действия	№ перехода
			4
4	проход по всем подчиненным объекта	инициализация указателя object типа base_class* значением результата метода search_in_branch указателя child с аргументом name	5
			6
5	object существует?	возврат object	∅
			4
6		возврат nullptr	∅

3.7 Алгоритм метода find класса base_class

Функционал: метод нахождения объекта.

Параметры: string coords - координаты объекта.

Возвращаемое значение: base_class* - указатель на найденный объект.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода find класса base_class

№	Предикат	Действия	№ перехода
1	coords пустая?	возврат nullptr	∅
		инициализация указателя parent типа base_class* на данный объект	2
2	первый символ строки coords равен '/'?		3
	первый символ строки coords равен '.'?		8
			10
3	результат метода get_parent указателя parent существует?	присваивание переменной parent значение результата метода get_parent указателя parent	3
			4

№	Предикат	Действия	№ перехода
4	значение строки coords равно "/"?	возврат parent	∅
			5
5	второй символ строки coords равен '/'?	удалить из coords первые два символа	6
			7
6		возврат результата метода search_in_tree указателя parent с аргументом coords	∅
7		удалить из coords первый символ	10
8	значение строки coords равно "."?	возврат parent	∅
		удалить из coords первый символ	9
9		возврат результата метода search_in_branch указателя parent с аргументом coords	∅
10		создание строковой переменной current	11
11		инициализация целочисленной переменной i значением 0	12
12	значение переменной i меньше размера строки coords?		13
		возврат результата метода get_child указателя parent с аргументом current	∅
13	i-тый символ строки coords равен '/'?	присвоить parent значение результата метода get_child указателя parent с аргументом current	14
		добавить в конец cuurent элемент coords с индексом i	16
14	parent не существует?	возврат parent	∅
			15
15		присваивание переменной current значене ""	16

№	Предикат	Действия	№ перехода
16		i++	12

3.8 Алгоритм функции count

Функционал: лямбда-функция для подсчета количества объектов с именем name.

Параметры: string name - имя объекта для подсчета, base_class* object - указатель на объект, у которого идет подсчет.

Возвращаемое значение: int - количетсво объектов с именем, заданным в параметре.

Алгоритм функции представлен в таблице 9.

Таблица 9 – Алгоритм функции count

№	Предикат	Действия	№ перехода
1		инициализация переменной с типа int значением 0	2
2	значение результата метода get_name указателя object равно значению параметра name?	c++	3
			3
3		инициализация переменной i типа int значением 0	4
4	значение переменной i меньше длины списка childrent указателя object?	увеличение значения переменной с на результат выполнения функции count с параметрами name и object->children[i]	5
		возврат c	∅
5		i++	4

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-17.

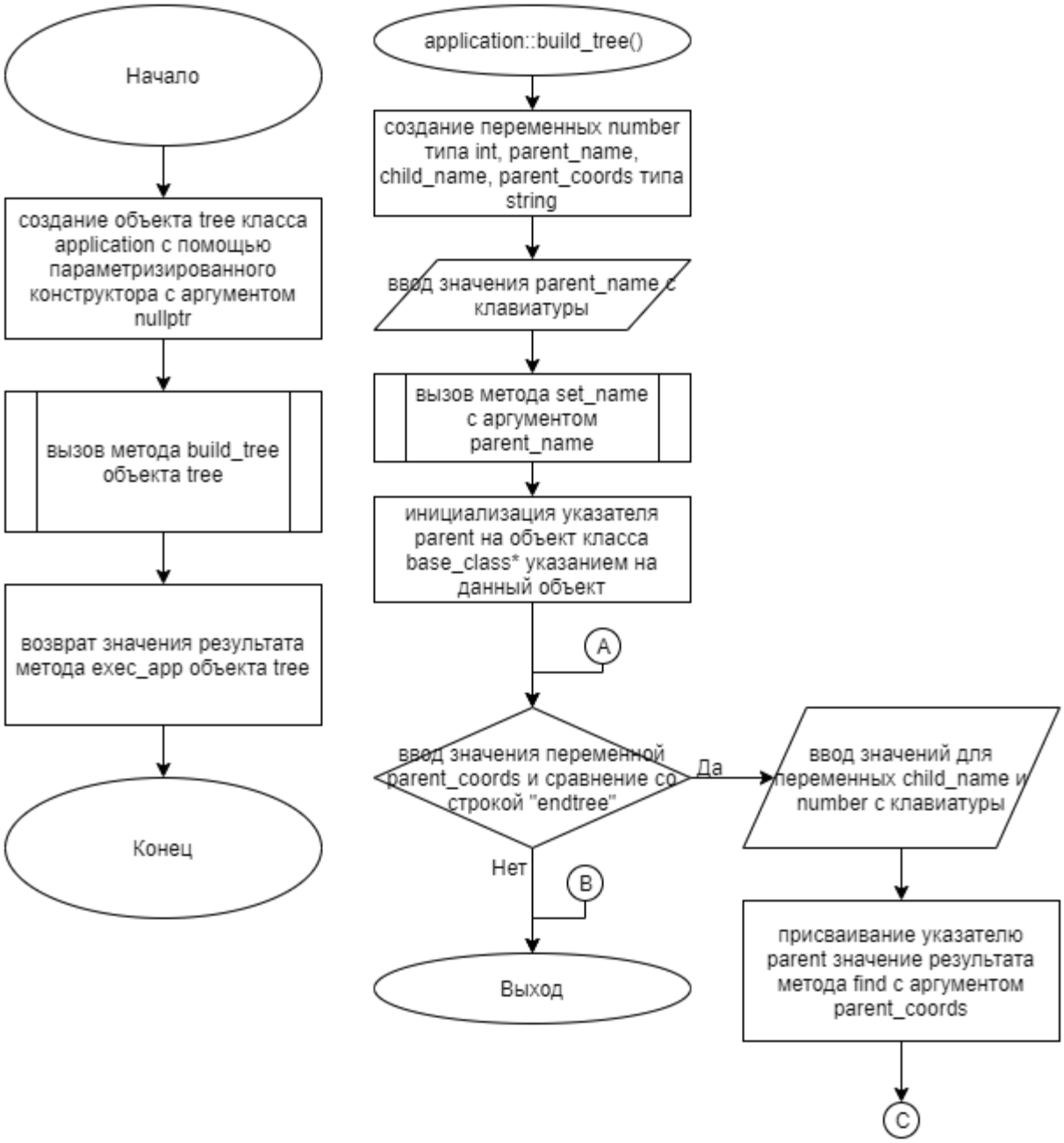

Рисунок 1 – Блок-схема алгоритма

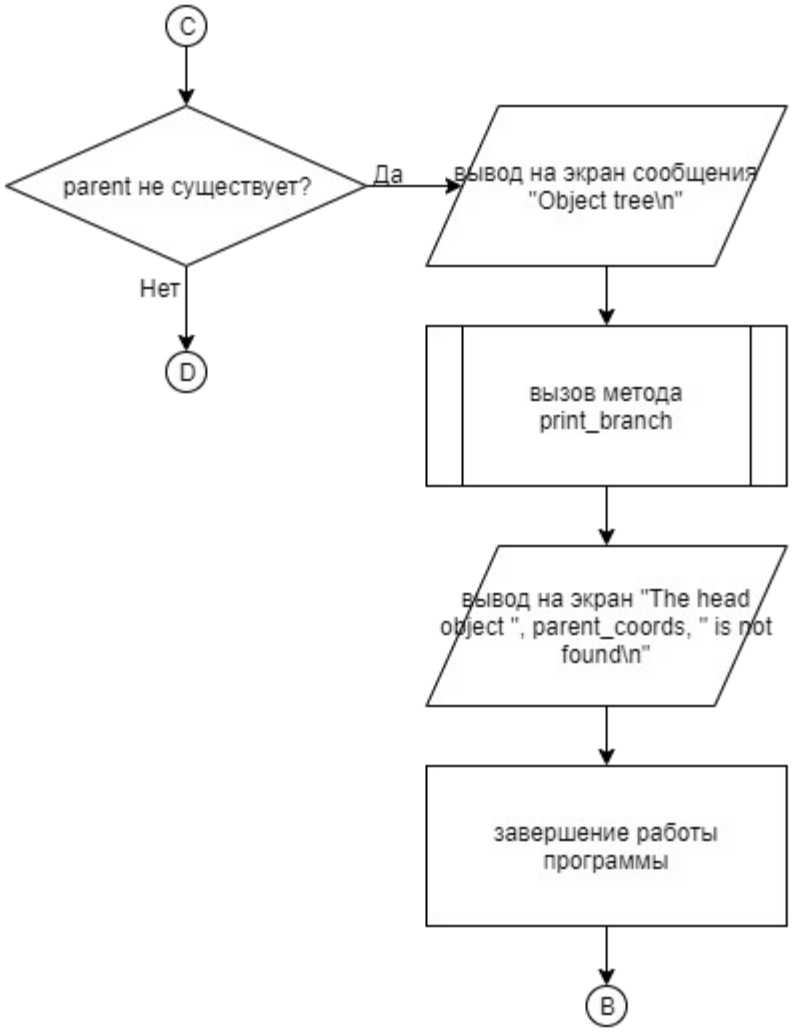

Рисунок 2 – Блок-схема алгоритма

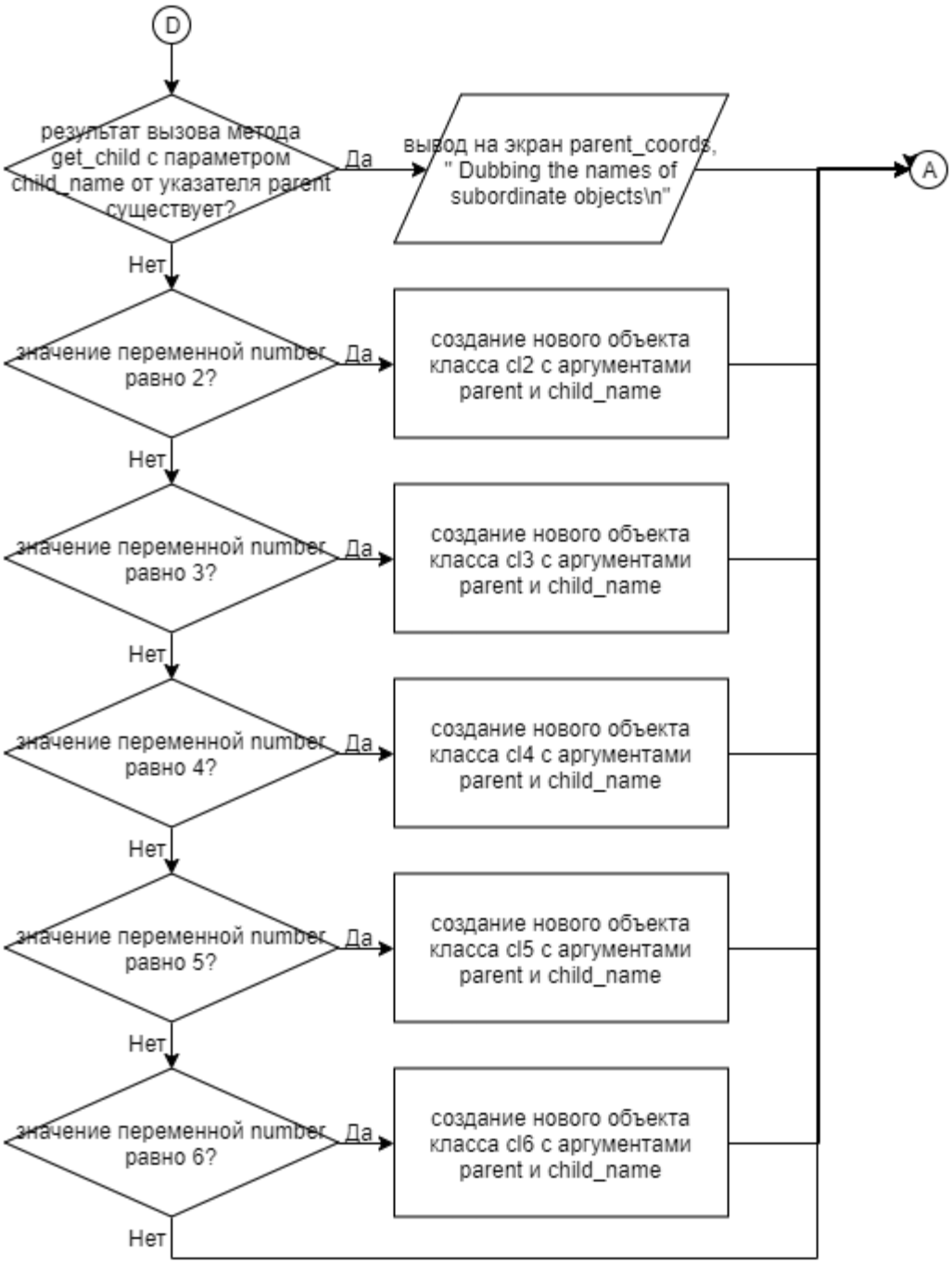

Рисунок 3 – Блок-схема алгоритма

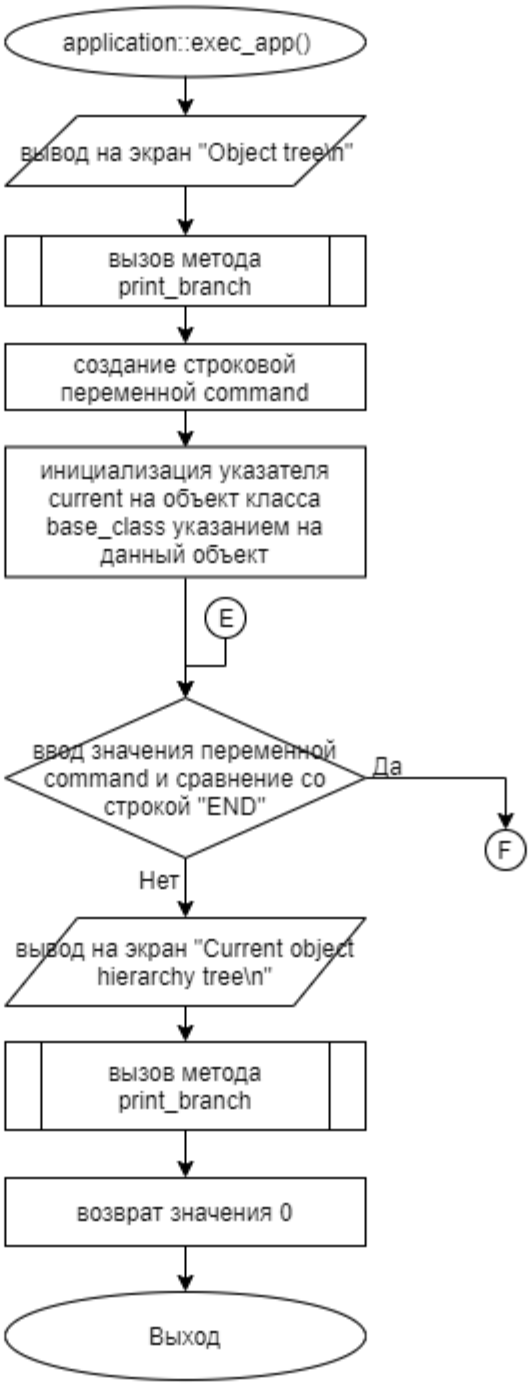

Рисунок 4 – Блок-схема алгоритма

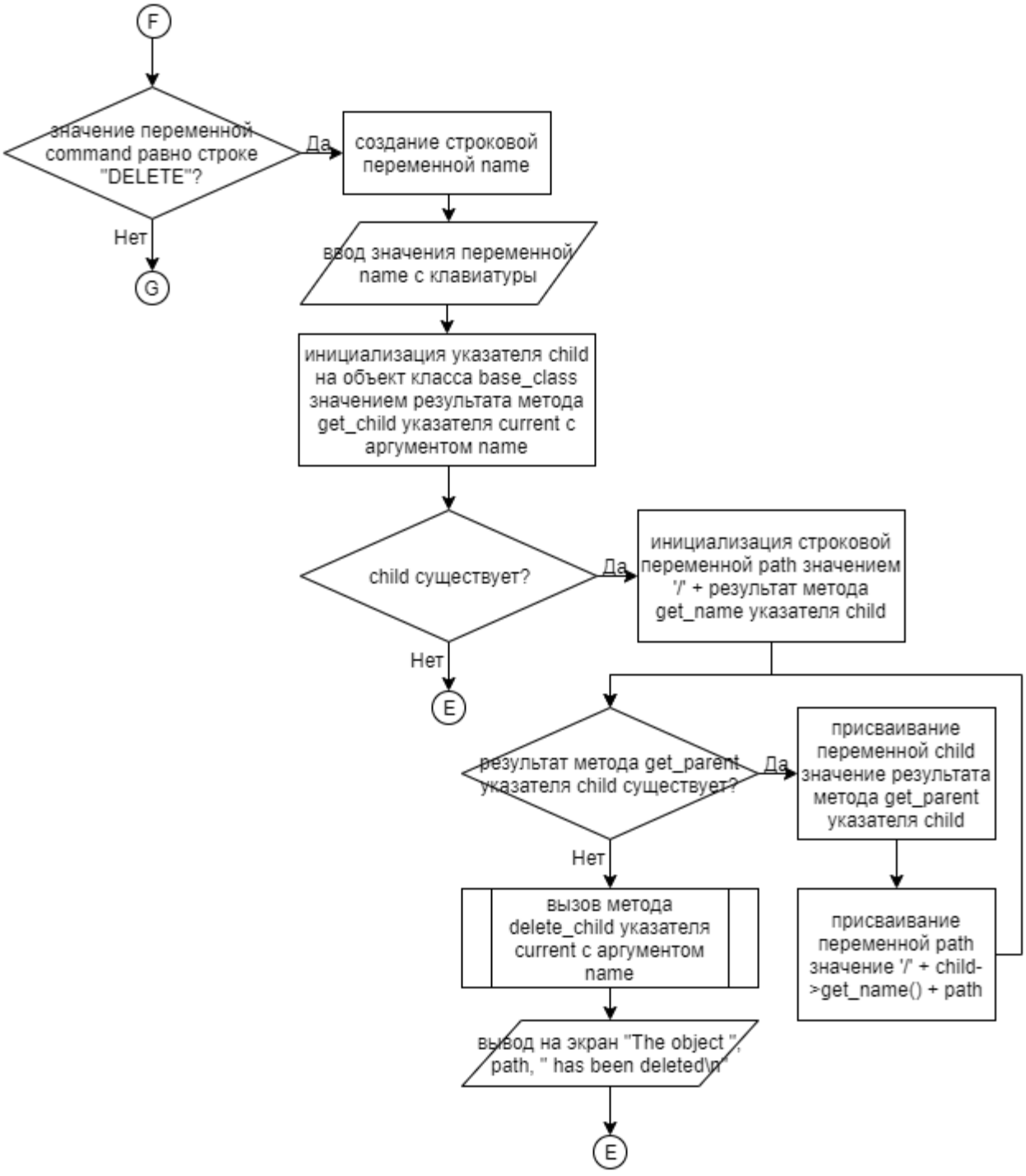

Рисунок 5 – Блок-схема алгоритма

Рисунок 6 – Блок-схема алгоритма

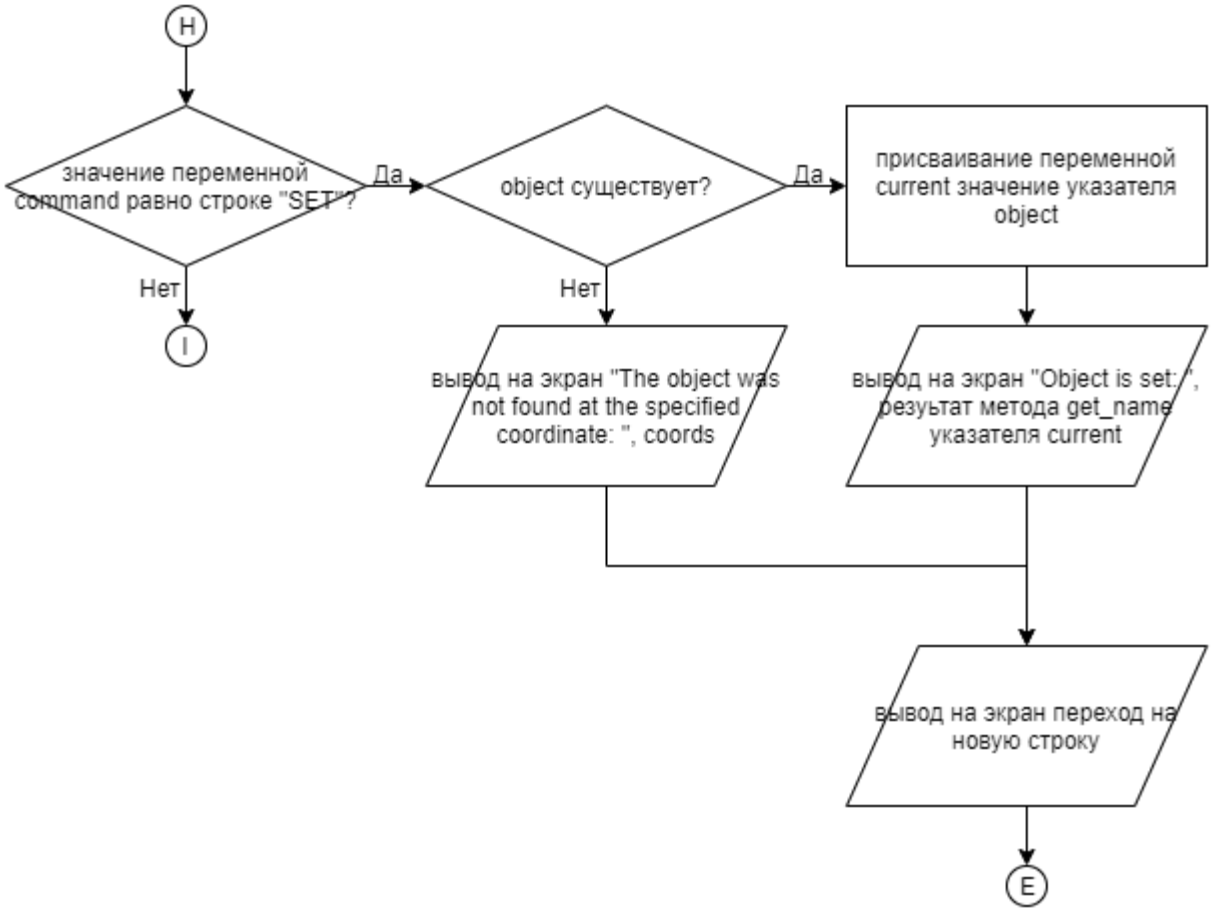

Рисунок 7 – Блок-схема алгоритма

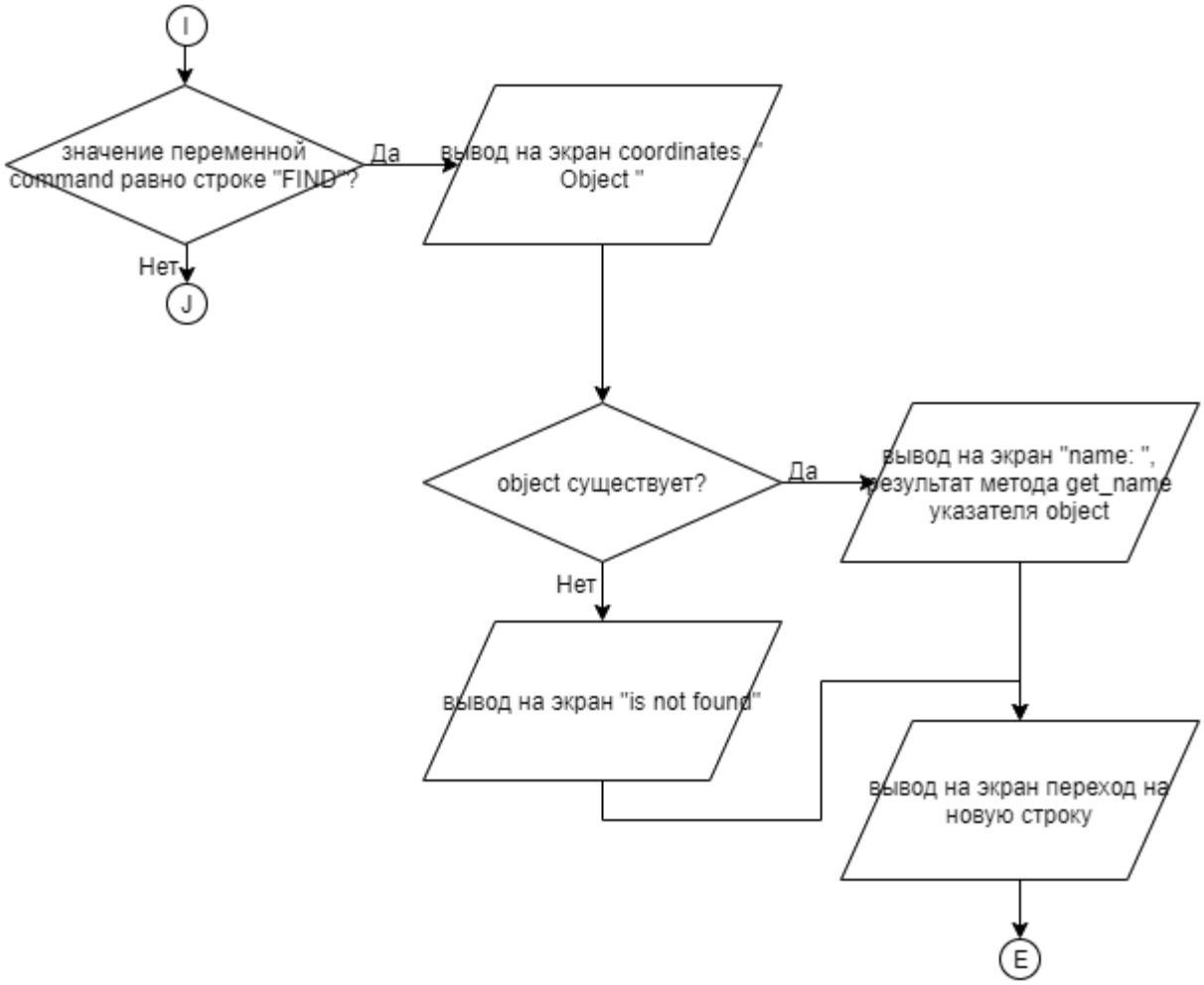

Рисунок 8 – Блок-схема алгоритма

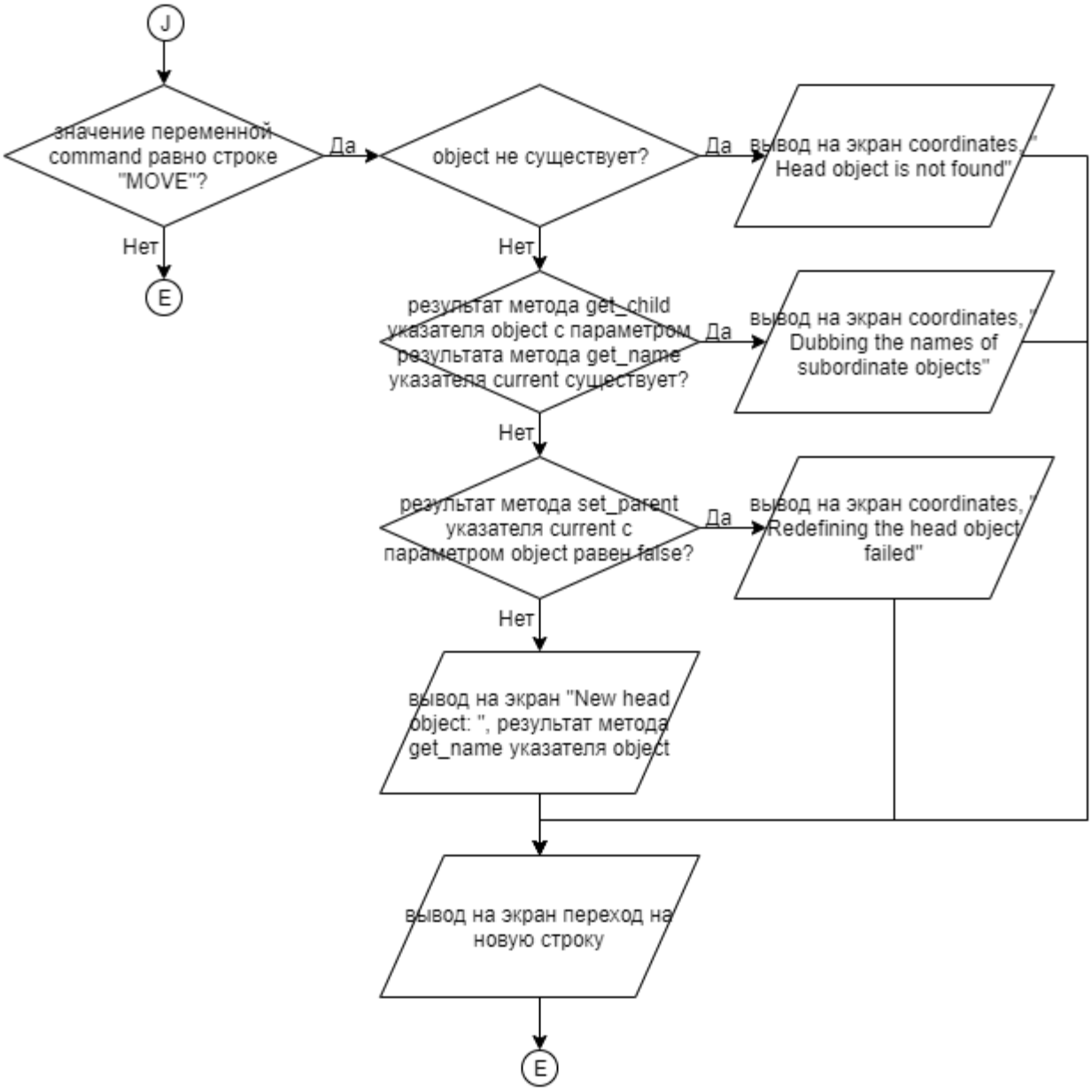

Рисунок 9 – Блок-схема алгоритма

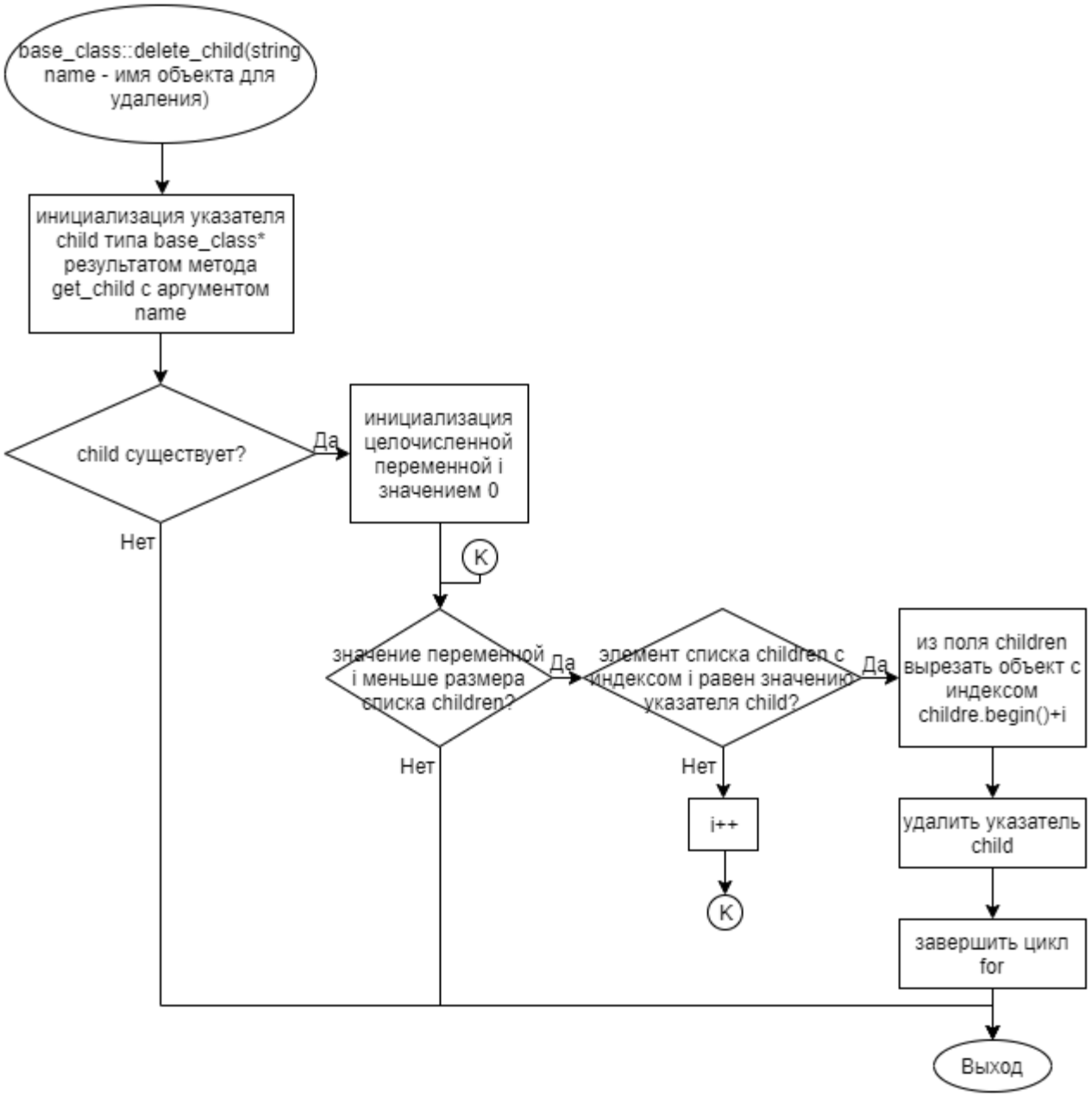

Рисунок 10 – Блок-схема алгоритма

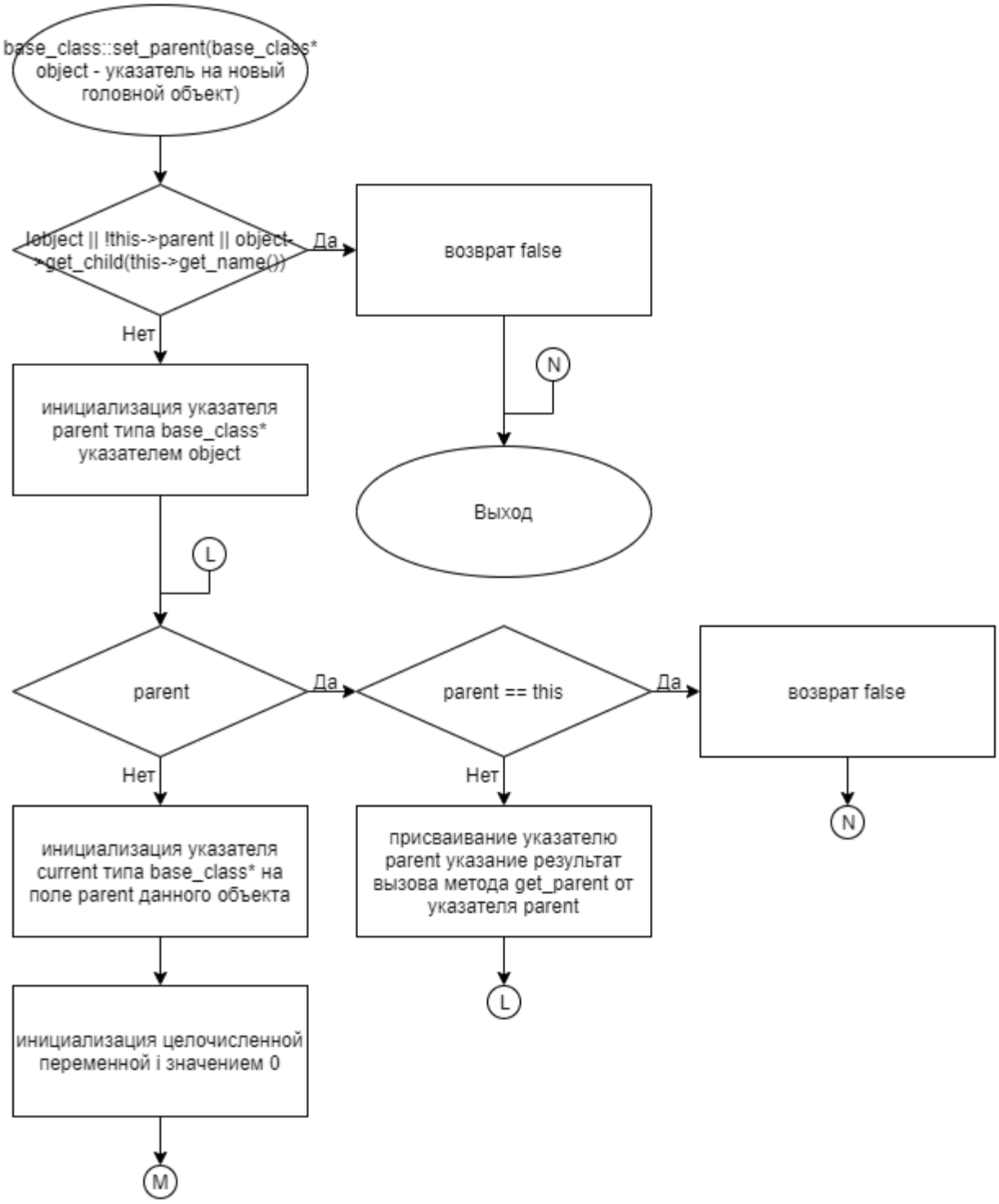

Рисунок 11 – Блок-схема алгоритма

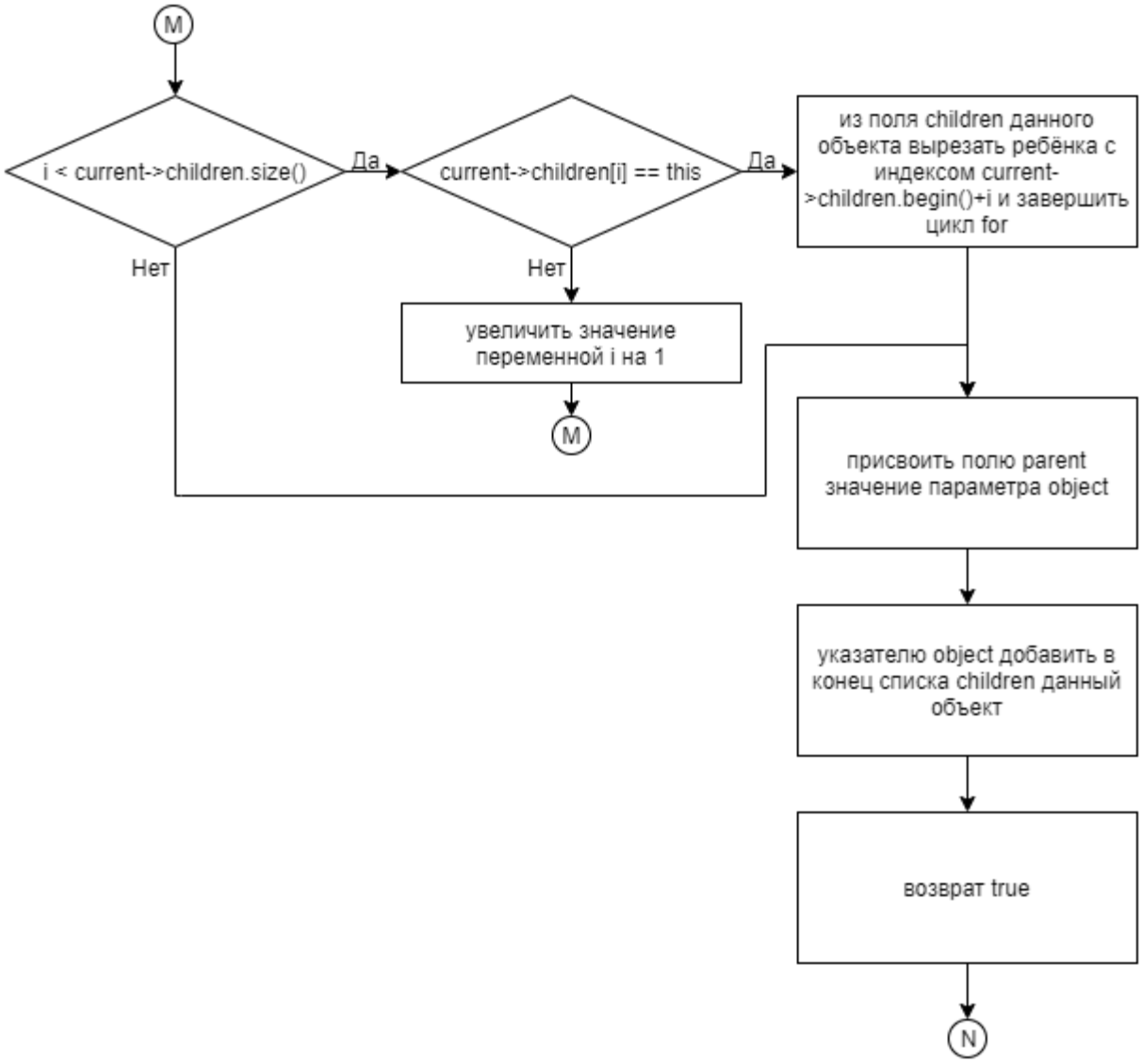

Рисунок 12 – Блок-схема алгоритма

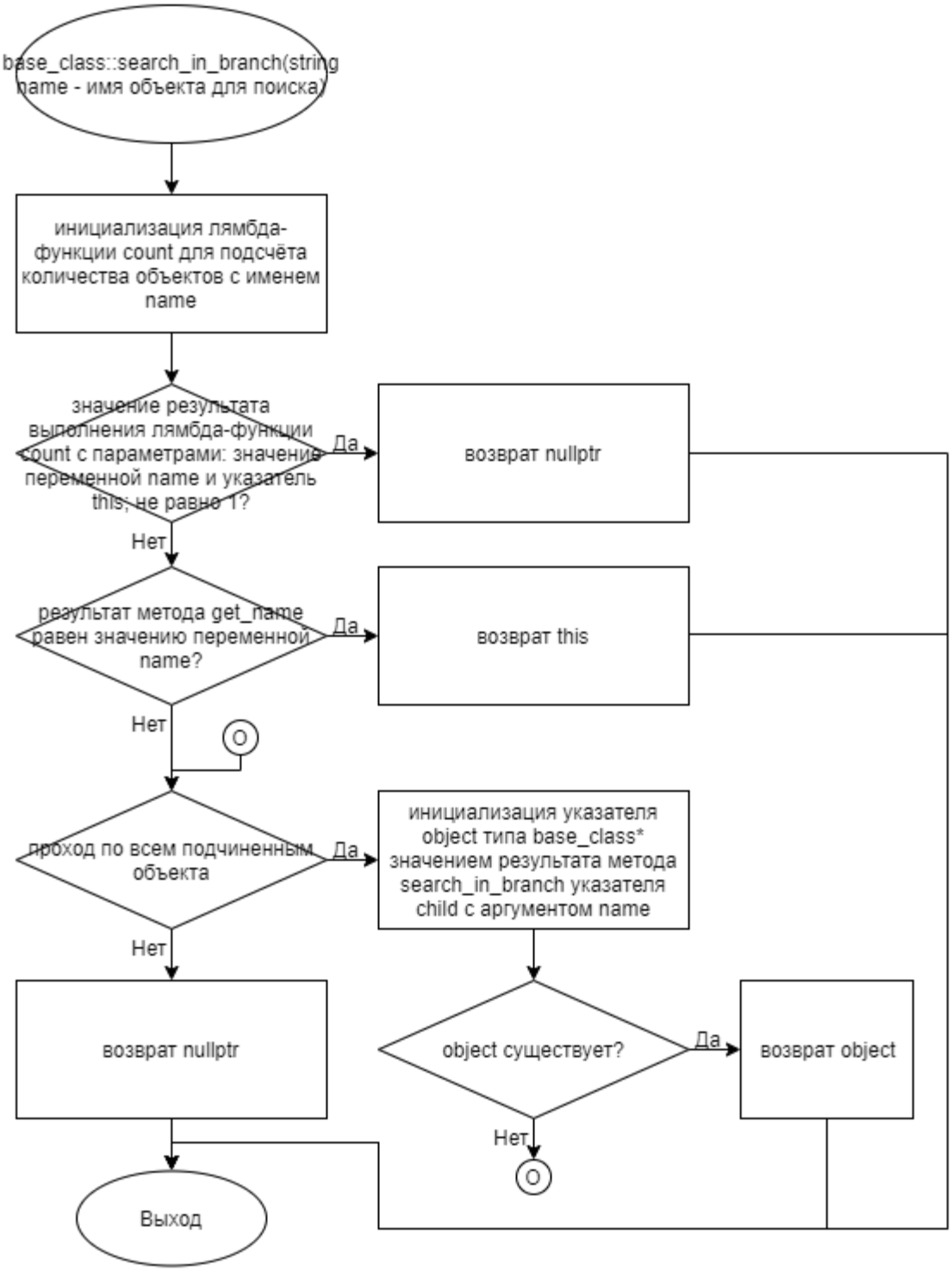

Рисунок 13 – Блок-схема алгоритма

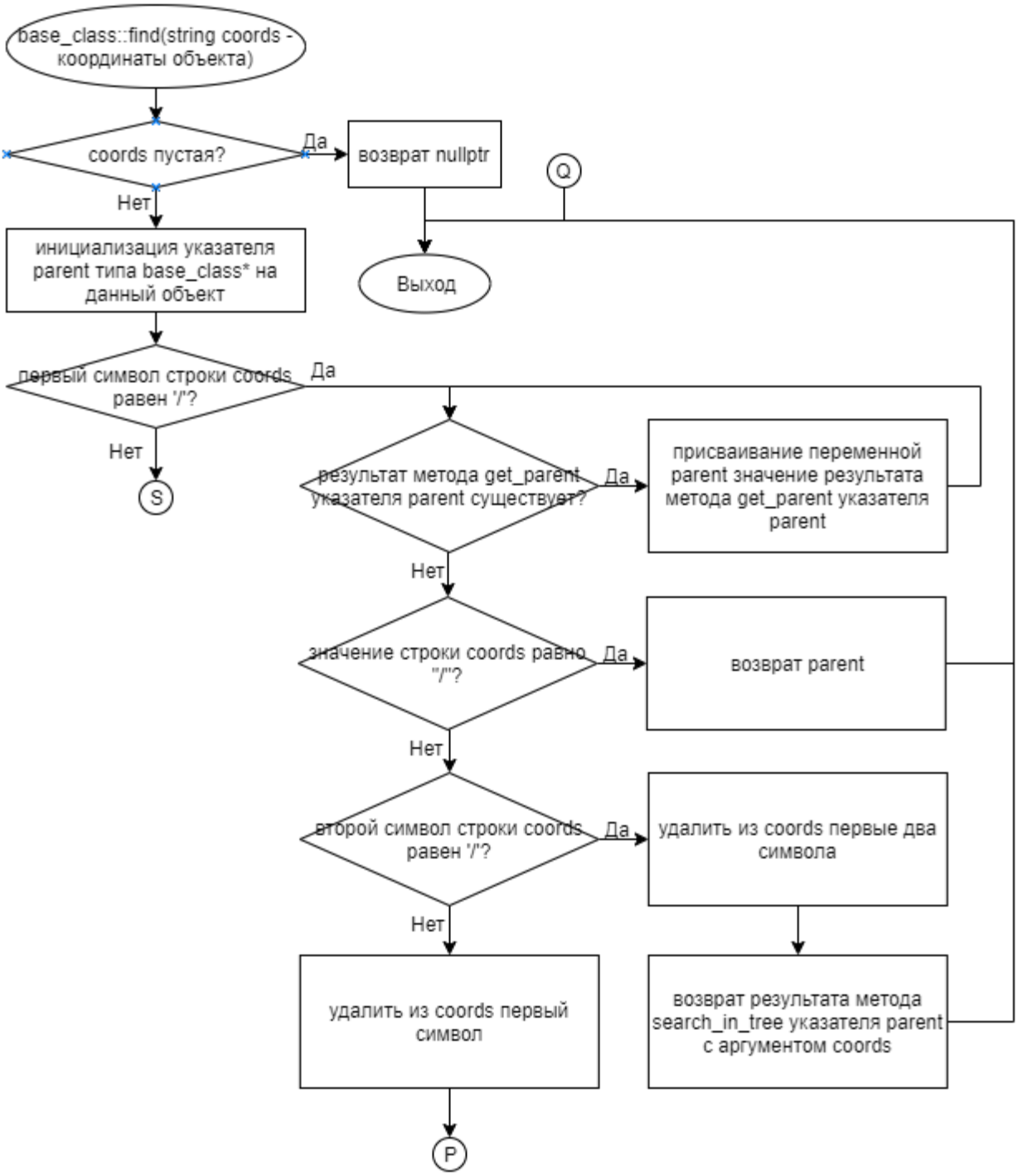

Рисунок 14 – Блок-схема алгоритма

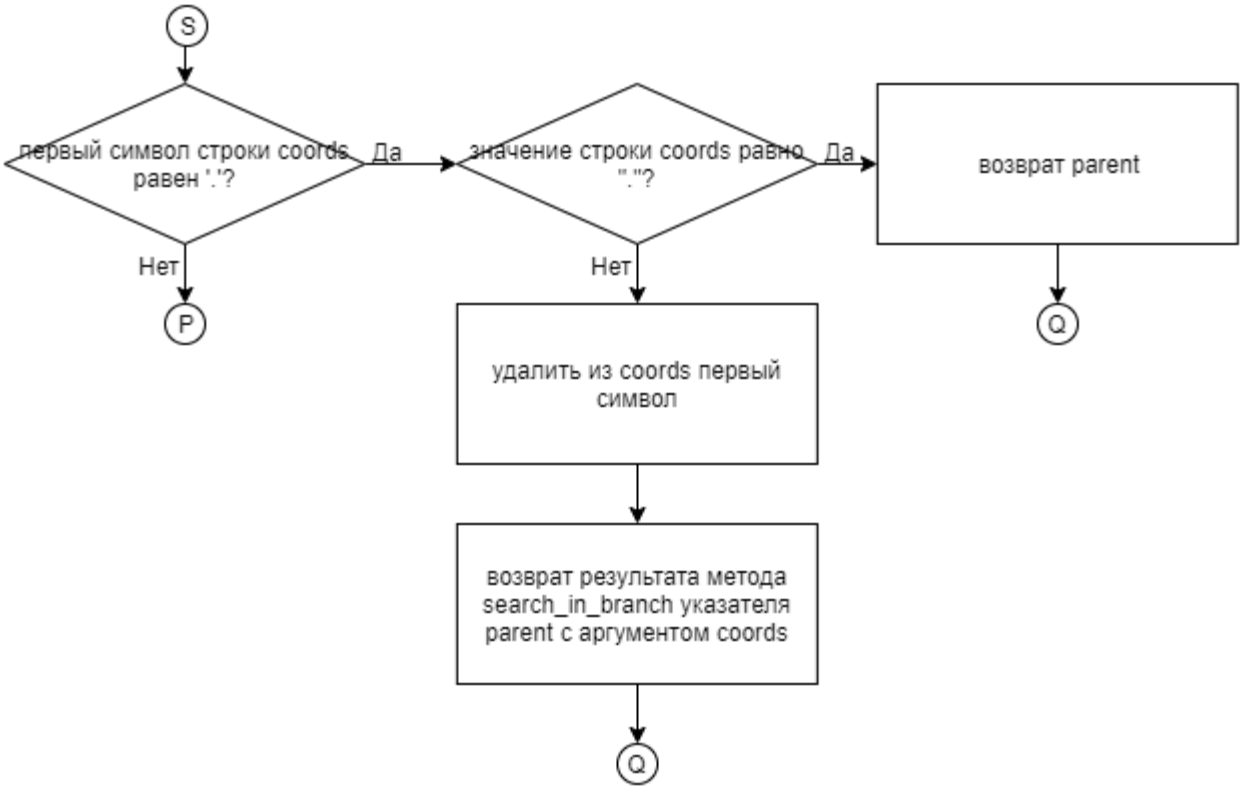

Рисунок 15 – Блок-схема алгоритма

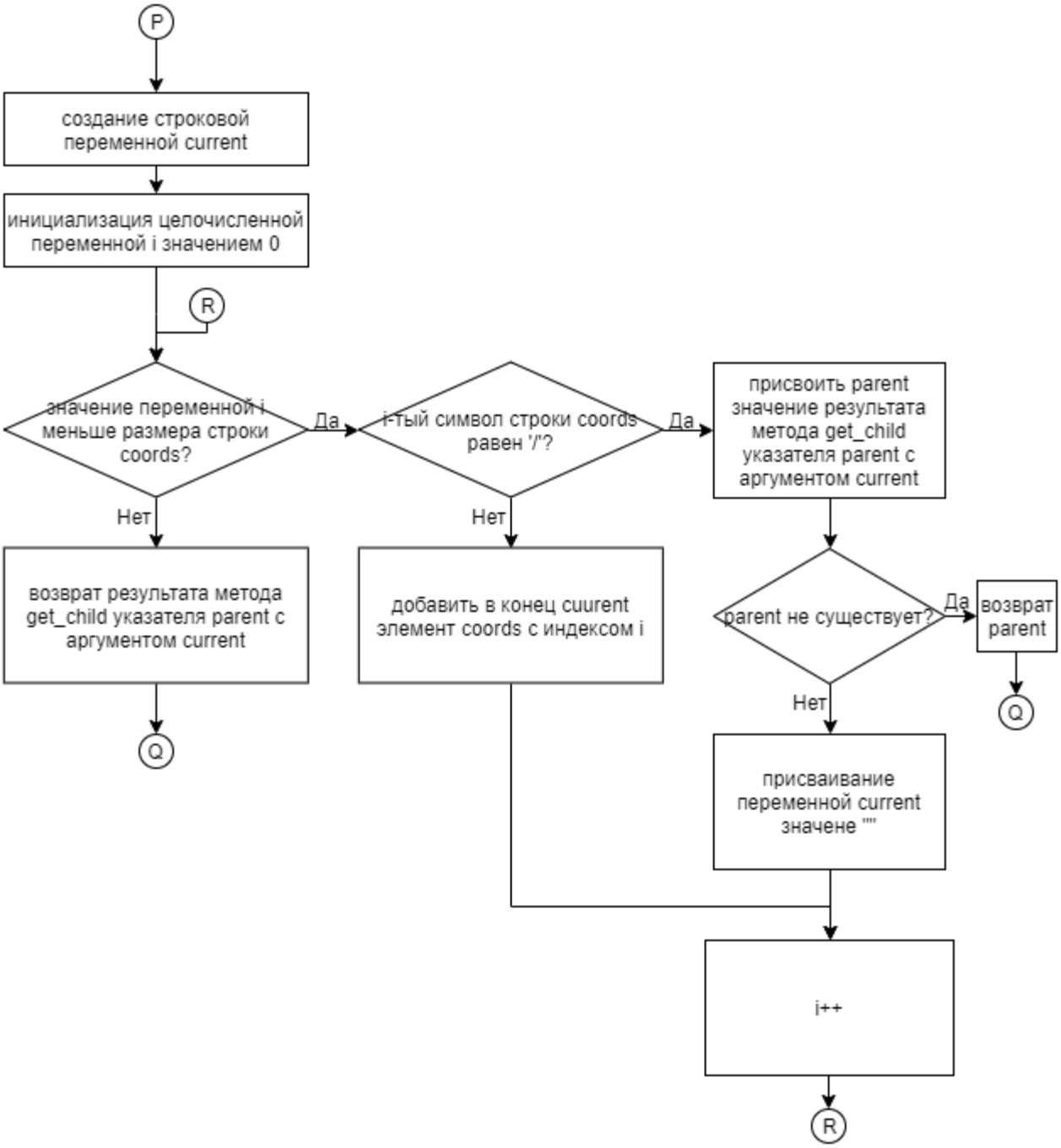

Рисунок 16 – Блок-схема алгоритма

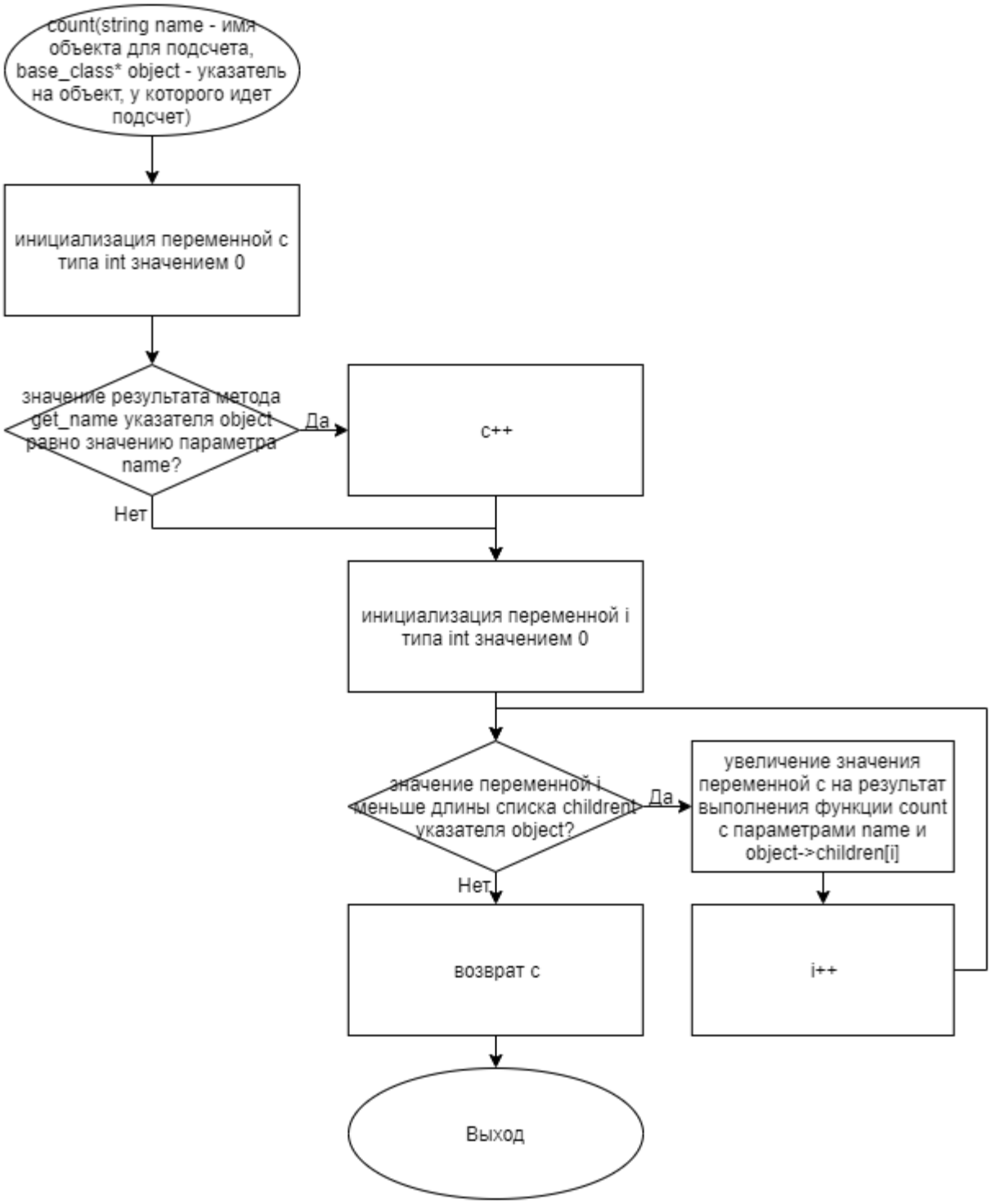

Рисунок 17 – Блок-схема алгоритма

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, parent_coords;
   std::cin >> parent_name;
   set_name(parent_name);
   base_class* parent = this;
   while (std::cin >> parent_coords && parent_coords != "endtree")
   {
      std::cin >> child_name >> number;
      parent = parent->find(parent_coords);
      if (!parent)
      {
         std::cout << "Object tree\n";
         print_branch();
         std::cout << "The head object " << parent_coords << " is not found\
n";
         exit(1);
      }
      if (parent->get_child(child_name))
         std::cout << parent_coords << "     Dubbing the names of subordinate
objects\n";
      else
      {
         if (number == 2)
            new cl2(parent, child_name);
         else if (number == 3)
            new cl3(parent, child_name);
```

```
            else if (number == 4)
               new cl4(parent, child_name);
            else if (number == 5)
               new cl5(parent, child_name);
            else if (number == 6)
               new cl6(parent, child_name);
         }
      }
   }
   int application::exec_app()
   {
      std::cout << "Object tree\n";
      print_branch();
      std::string command;
      base_class* current = this;
      while (std::cin >> command && command != "END")
      {
         if (command == "DELETE")
         {
            std::string name;
            std::cin >> name;
            base_class* child = current->get_child(name);
            if (child)
            {
               std::string path = '/' + child->get_name();
               while (child->get_parent() != this)
               {
                  child = child->get_parent();
                  path = '/' + child->get_name() + path;
               }
               current->delete_child(name);
               std::cout <<"The object " << path << " has been deleted\n";
            }
         }
         else if (command == "SET" || command == "FIND" || command == "MOVE")
         {
            std::string coords;
            std::cin >> coords;
            base_class* object = current->find(coords);
            if (command == "SET")
            {
               if (object)
               {
                  current = object;
                  std::cout << "Object is set: " << current->get_name();
               }
               else
                  std::cout  <<  "The  object  was  not  found  at  the  specified
   coordinate: " << coords;
               std::cout << std::endl;
            }
            else if (command == "FIND")
            {
               std::cout << coords << "     Object ";
               if (object) std::cout << "name: " << object->get_name();
```

```
            else std::cout << "is not found";
            std::cout << std::endl;
         }
         else if (command == "MOVE")
         {
            if (!object)
               std::cout << coords << "     Head object is not found";
            else if (object->get_child(current->get_name()))
               std::cout << coords << "     Dubbing the names of subordinate
objects";
            else if (!current->set_parent(object))
               std::cout  <<  coords  <<  "        Redefining  the  head  object
failed";
            else
               std::cout << "New head object: " << object->get_name();
            std::cout << std::endl;
         }
      }
   }
   std::cout << "Current object hierarchy tree\n";
   print_branch();
   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(); //changed
};

#endif
```

5.3 Файл base_class.cpp

Листинг 3 – base_class.cpp

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

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;
}

base_class* base_class::search_in_branch(std::string name) //changed
{
   std::function <int(std::string name, base_class* object)> count = [&count]
(std::string name, base_class* object)
   {
      int c = 0;
      if (object->get_name() == name) c++;
      for (int i = 0; i < object->children.size(); i++)
         c += count(name, object->children[i]);
      return c;
   };
   if (count(name, this) != 1) return nullptr;
   if (get_name() == name) return this;
   else
      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::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();
}

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;
   }
}

base_class* base_class::find(std::string coords) //new
{
   if (coords.empty()) return nullptr;
   base_class* parent = this;
   if (coords[0] == '.')
   {
      if (coords == ".")
         return this;
      coords.erase(0, 1);
      return this->search_in_branch(coords);
   }
   else if (coords[0] == '/')
   {
      while (parent->get_parent())
         parent = parent->get_parent();
      if (coords == "/")
         return parent;
      if (coords[1] == '/')
      {
         coords.erase(0, 2);
         return parent->search_in_tree(coords);
      }
      coords.erase(0, 1);
   }
   std::string current = "";
   for (int i = 0; i < coords.length(); i++)
   {
      if (coords[i] == '/')
      {
         parent = parent->get_child(current);
         if (!parent) return parent;
         current = "";
      }
      else current += coords[i];
   }
   return parent->get_child(current);
}
void base_class::delete_child(std::string name) //new
{
   base_class* child = get_child(name);
   if (child)
      for (int i = 0; i < children.size(); i++)
         if (children[i] == child)
         {
            children.erase(children.begin() + i);
            delete child;
            break;
```

```
        }
}
bool base_class::set_parent(base_class* object) //new
{
   if  (!object  ||  !this->parent  ||  object->get_child(this->get_name()))
return false;
   base_class* parent = object;
   while (parent)
   {
      if (parent == this) return false;
      parent = parent->get_parent();
   }
   base_class* current = this->parent;
   for (int i = 0; i < current->children.size(); i++)
      if (current->children[i] == this)
      {
         current->children.erase(current->children.begin() + i);
         break;
      }
   this->parent = object;
   object->children.push_back(this);
   return true;
}
```

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;
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);
   base_class* search_in_branch(std::string name); //changed
   base_class* search_in_tree(std::string name);
```

```
        void print_branch();
        void print_branch_status();
        void set_status(int status);
        base_class* find(std::string coords); //new
        void delete_child(std::string name); //new
        bool set_parent(base_class* object); //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 10.

Таблица 10 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

rootela
/ 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_7
FIND
object_4/object_7
FIND .
FIND .object_7
FIND
object_4/object_7
MOVE .object_7
SET
object_4/object_7
MOVE //object_1
MOVE /object_3
END
	Object tree
	rootela
	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_7
	Object is not found
	object_4/object_7
	Object name:
	object_7
	. Object name:
	object_2
	.object_7 Object
	name: object_7
	object_4/object_7
	Object name:
	object_7
	.object_7
	Redefining the head
	object failed
	Object is set:
	object_7
	//object_1
	Dubbing the names of
	subordinate objects
	New head object:
	object_3
	Current object
	hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_5
		Object tree
		rootela
		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_7
		Object is not found
		object_4/object_7
		Object name:
		object_7
		. Object name:
		object_2
		.object_7 Object
		name: object_7
		object_4/object_7
		Object name:
		object_7
		.object_7
		Redefining the head
		object failed
		Object is set:
		object_7
		//object_1
		Dubbing the names of
		subordinate objects
		New head object:
		object_3
		Current object
		hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_5

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_3 object_3 object_7	object_3 object_3 object_7

rootela
/ 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_7
FIND
object_4/object_7
FIND .
FIND .object_7
FIND
object_4/object_7
MOVE .object_7
SET
object_4/object_7
MOVE //object_1
MOVE /object_3
SET /
DELETE object_3
END
	Object tree
	rootela
	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_7
	Object is not found
	object_4/object_7
	Object name:
	object_7
	. Object name:
	object_2
	.object_7 Object
	name: object_7
	object_4/object_7
	Object name:
	object_7
	.object_7
	Redefining the head
	object failed
	Object is set:
	object_7
	//object_1
	Dubbing the names of
	subordinate objects
	New head object:
	object_3
	Object is set:
	rootela
	The object /object_3
	has been deleted
	Current object
	hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
		Object tree
		rootela
		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_7
		Object is not found
		object_4/object_7
		Object name:
		object_7
		. Object name:
		object_2
		.object_7 Object
		name: object_7
		object_4/object_7
		Object name:
		object_7
		.object_7
		Redefining the head
		object failed
		Object is set:
		object_7
		//object_1
		Dubbing the names of
		subordinate objects
		New head object:
		object_3
		Object is set:
		rootela
		The object /object_3
		has been deleted
		Current object
		hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_5 object_3	object_5 object_3
root / obj_1 3 / obj_2 2 /obj_2 obj_4 3 / obj_3 3 /obj_2 obj_3 6 endtree FIND obj_2/obj_4 SET /obj_2 FIND . MOVE //obj_1 MOVE /obj_3 END	Object tree root obj_1 obj_2 obj_4 obj_3 obj_3 obj_2/obj_4 Object name: obj_4 Object is set: obj_2 . Object name: obj_2 New head object: obj_1 New head object: obj_3 Current object hierarchy tree root obj_1 obj_3 obj_2 obj_4 obj_3	Object tree root obj_1 obj_2 obj_4 obj_3 obj_3 obj_2/obj_4 Object name: obj_4 Object is set: obj_2 . Object name: obj_2 New head object: obj_1 New head object: obj_3 Current object hierarchy tree root obj_1 obj_3 obj_2 obj_4 obj_3
rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 SET /object_2 FIND //object_7 FIND object_4/object_7 SET / DELETE object_3 END	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_2/object_4 Object name: object_4 Object is set: object_2 //object_7 Object is not found object_4/object_7 Object name: object_7 Object is set: rootela Current object hierarchy tree rootela object_1 object_7	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_2/object_4 Object name: object_4 Object is set: object_2 //object_7 Object is not found object_4/object_7 Object name: object_7 Object is set: rootela Current object hierarchy tree rootela object_1 object_7

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_2 object_4 object_7	object_2 object_4 object_7
rootela / object_1 3 / object_2 2 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 endtree FIND object_2/object_4 SET /object_2 FIND //object_7 FIND object_4/object_7 DELETE object_6 MOVE /object_2 SET object_4/object_7 MOVE //object_1 END	Object tree rootela object_1 object_7 object_2 object_5 object_3 object_3 object_2/object_4 Object is not found Object is set: object_2 //object_7 Object name: object_7 object_4/object_7 Object is not found /object_2 Redefining the head object failed The object was not found at the specified coordinate: object_4/object_7 New head object: object_1 Current object hierarchy tree rootela object_1 object_7 object_2 object_5 object_3 object_3	Object tree rootela object_1 object_7 object_2 object_5 object_3 object_3 object_2/object_4 Object is not found Object is set: object_2 //object_7 Object name: object_7 object_4/object_7 Object is not found /object_2 Redefining the head object failed The object was not found at the specified coordinate: object_4/object_7 New head object: object_1 Current object hierarchy tree rootela object_1 object_7 object_2 object_5 object_3 object_3

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).